



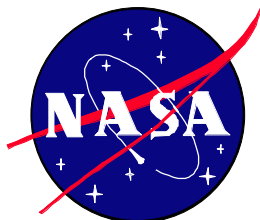
Guidelines for the Rapid Development of Software Systems

— References —

ENGINEERING DIRECTORATE

AEROSCIENCE AND FLIGHT MECHANICS DIVISION

2 December 1996



**National Aeronautics and
Space Administration**

**Lyndon B. Johnson Space Center
Houston, TX**



Guidelines for the Rapid Development of Software Systems

— References —

Prepared By:

Denise M. DiFilippo
G. B. Tech, Incorporated

Bill G. Brown
SysComm Development

Approved By:

David A. Petri
GN&C Rapid Development Lab Manager
Aeroscience and Flight Mechanics Division
NASA/Johnson Space Center

James P. Ledet
Code Q RTOP Project Manager
Aeroscience and Flight Mechanics Division
NASA/Johnson Space Center

Concurred By:

Aldo J. Bordano, Chief
Aeroscience and Flight Mechanics Division
NASA/Johnson Space Center

Sonya F. Sepahban, Deputy Chief
Aeroscience and Flight Mechanics Division
NASA/Johnson Space Center

Preface

This document (JSC 38606) contains reference material that was cited in a companion document (JSC 38605) and that is not commonly available in technical libraries. Primarily, these are internal working papers related to the study subject. Together the documents represent the results of work performed in FY'96 with funds provided under the Research and Technology Operation Plan (RTOP) by the Office of Safety and Mission Assurance (OSMA). OSMA has delegated requirements for the Agency Software Program to Ames Research Center Software Technology Division (ARC/IT) located in Fairmont, West Virginia. Work under this initiative was managed at ARC/IT by Kathryn M. Kemp, Deputy Chief, Software Technology Division, and George J. Sabolish, Center Software Initiative Manager. The work was performed in the Aeroscience and Flight Mechanics Division at the Johnson Space Center in collaboration with the Jet Propulsion Laboratory.

The results of FY'96 work are documented in a 2 volume set consisting of:

- JSC 38605 Guidelines for the Rapid Development of Software Systems
- JSC 38606 Guidelines for the Rapid Development of Software Systems - References

This initiative continues in FY'97 with the objective of determining the effectiveness of the guidelines by using them in a rapid software development demonstration project. The results of the demonstration project will be documented along with any refinement to these guidelines.

Table of Contents

1. J.M. Ball & E.J. Riel, "Strategic Avionics Technology Program: Rapid Prototyping & Integrated Design Application Studies, Final Report"; NAS9-18877, February 1995.
 2. J.M. Ball, D.C. Weed & E.J. Riel, "Strategic Avionics Technology Program: Rapid Prototyping & Integrated Design Application Studies, White Paper, Rapid Development Process"; NAS9-18877, February 1995.
 3. D. M. DiFilippo, "Aeroscience & Flight Mechanics Division (AFMD) Guidance, Navigation & Control (GN&C) Rapid Development Laboratory Processes: Historical Perspective"; McDonnell Douglas TM-960030-03, May 1, 1996.
 4. D. Pesek, "Rapid Development Lab Configuration Management Plan"; internal document, October 1995.
 5. Rapid Prototyping Simulation Development Team, "Soyuz ACRV Simulation Development Progress Report, April-May 1993"; McDonnell Douglas TM-6.23.07-24; June 30, 1993.
 6. Rapid Prototyping Simulation Development Team, "Soyuz ACRV Simulation Development Lessons Learned Report"; McDonnell Douglas TM-0009-01 enclosure 1; January 28, 1994.
 7. Rapid Prototyping Simulation Development Team, "Soyuz ACRV Simulation Development Assurance and Test Report"; McDonnell Douglas TM-0009-01 enclosure 2; January 28, 1994
 8. Rapid Prototyping Simulation Development Team, "Soyuz ACRV Simulation Development Configuration Management Plan"; McDonnell Douglas TM-0009-01 enclosure 3; January 28, 1994
 9. Rapid Prototyping Simulation Development Team, "Soyuz ACRV Simulation Development Trip Report, Summary of Trip to MDA-Huntington Beach"; McDonnell Douglas TM-0009-01 enclosure 4; January 28, 1994
 10. J. Uhde & D. Weed, "Library Reuse in a Rapid Development Environment"; *Proceedings of the AIAA Conference on Computing & Aerospace X*, March 28-30, 1995, San Antonio, TX; pp. 521-530.
 11. J. Uhde-Lacovara, D. Weed, B. McCleary, & R. Wood, "The Rapid Development Process Applied to Soyuz Simulation Production", internal document, 1994
-

**Aeroscience & Flight Mechanics Division (AFMD) Guidance, Navigation
& Control (GN&C) Rapid Development Laboratory Processes:
Historical Perspective**

01 May 1996

D. M. DiFilippo

Process Definition for Rapid Development of Software

**McDonnell Douglas Aerospace
Space and Defense Systems - Houston Division
13100 Space Center Blvd.
Houston, Texas 77059-3556**

**NATIONAL AERONAUTICS AND SPACE ADMINISTRATION
JOHNSON SPACE CENTER**

**In Accordance with NASA Contract NAS9-19100
Subcontract 02C0100001**

Delivered to NASA in MDA-HD Transmittal Memo TM-960030-03

Table of Contents

<i>Section</i>		<i>page</i>
1.0	Introduction.....	1
2.0	The Rapid Software Development Paradigm	1
3.0	The JSC GN&C Rapid Development Laboratory	3
4.0	The Soyuz Assured Crew Return Vehicle Simulation.....	5
5.0	Rapid Software Development Results and Lessons Learned	6
6.0	Topics for Further Study	7
7.0	Early Thoughts and Special Concerns	8
8.0	RTOP Plans and Schedule.....	10
9.0	Related Documents.....	10

Figures

Figure		Page
1	The Rapid Development Process.....	2
2	The Design Team Concept	3
3	Soyuz Simulation Development Model.....	6

Tables

Table		Page
1	Software Tools in the RDL.....	4
2	Hardware Tools in the RDL.....	4
3	Soyuz Simulation Phase 1 and 2 Metrics	7

Acronyms and Abbreviations

ACRV	Assured Crew Return Vehicle
AFMD	Aeroscience and Flight Mechanics Division
COTS	commercial, off-the-shelf
CMM	Capability Maturity Model
DOF	degree-of-freedom
GN&C	Guidance, Navigation & Control
HIL	hardware-in-the-loop
ISI	Integrated Systems Inc.
ISSA	International Space Station Alpha
JPL	Jet Propulsion Laboratory
JSC	Johnson Spaceflight Center
MDA-HD	McDonnell Douglas Aerospace – Houston Division
RDL	Rapid Development Laboratory
RTOP	Research and Technology Objectives and Plan
SEI	Software Engineering Institute

1.0 Introduction

The Aeroscience and Flight Mechanics Division (AFMD) at the National Aeronautics and Space Administration-Johnson Space Center (NASA-JSC) Engineering Directorate is exploring ways of producing Guidance, Navigation & Control (GN&C) systems more efficiently and effectively. A significant portion of this effort is software development, integration, testing and verification.

To achieve these goals, in the late 1980's AFMD established the GN&C Rapid Development Laboratory (RDL), a hardware/software facility designed to take a GN&C design project from initial inception through high-fidelity, real-time, hardware-in-the-loop (HIL) testing and perform final, end-to-end, GN&C system verification. The operations approach for the RDL concentrated on the use of commercial, off-the-shelf (COTS) software products to develop the GN&C algorithms in the form of graphical data flow diagrams, to automatically generate source code from these diagrams and to run in a real-time, HIL environment under a rapid development paradigm.

As an initial application of these concepts and tools, AFMD took on the development of a real-time, six degree-of-freedom (DOF) simulation of the Russian Soyuz vehicle. The processes and tools used in this effort, along with the lessons learned, were encouraging and, overall, support the premise of the RDL and the experiences and knowledge of the RDL team. That is, these new methodologies and tools can be applied to the development of simulation and flight software for complicated Guidance, Navigation & Control (GN&C) systems to improve quality, reduce the development time, reduce the cost, reduce project risk, or some combination of these.

Building on this initial work and on-going flight software development projects for X-vehicles, an RTOP (Research and Technology Objectives and Plan) was proposed and accepted to define and document generic, rigorous, reference processes and metrics for rapid development and integrated design and verification of flight software. This paper represents the kick-off point for the RTOP, describing the status of Rapid Application Development in GN&C at JSC. Some areas of special interest and initial questions, concerns and ideas are also presented.

2.0 The Rapid Software Development Paradigm

The rapid development process is a relatively new paradigm for hardware and software development that utilizes features of the spiral development model, rapid prototyping, and incremental development through the product development lifecycle. The process utilizes state-of-the-art computer-aided system engineering and software engineering tools which combine a graphical designer's environment with automatic software code generation and documentation.

The rapid development paradigm for software system development differs from the classic waterfall methodology in that it uses a spiral development process (Boehm, 1988). This is

characterized by a “build a little, test a little”, and in the GN&C environment “fly a little”, philosophy, with ample opportunity for feedback from and active participation of analysts, designers, programmers and users. The spiral development process accelerates system development through concurrent iteration on system requirements, design, code, unit test, and system integrated test issues and processes rather than the sequential execution of these processes found in the classic development methodology.

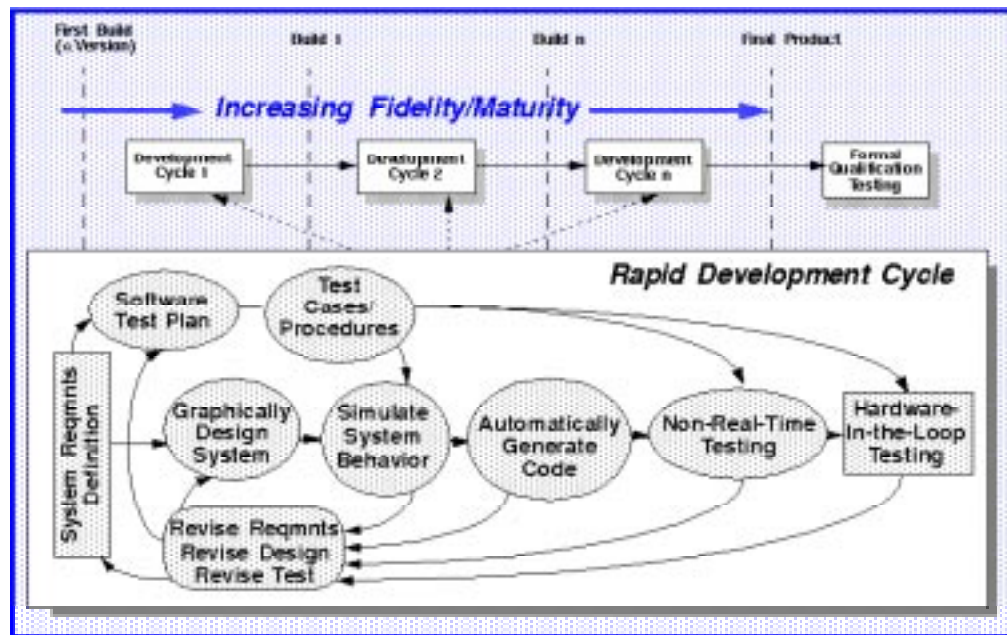


Figure 1. The Rapid Development Process

Rapid development projects are also characterized by the integration of all hardware and software elements early in the development cycle. This serves to identify requirements issues and system integration and connectivity problems early, so that they can be addressed and fixed at a phase in the development cycle which can accommodate the modifications with minimal impact and cost, resulting in lower risks.

This approach to software development is organized around a design team which assumes ownership of the entire development process and end products. Critical elements of the design team include a core development team and domain experts. The small core team (usually 1 to 5 people, depending on project size and complexity) integrates all project elements, provides configuration control functions, administers quality control processes, and ensures the project remains focused. Domain experts provide technical expertise across the range of technical disciplines in the project. The team structure contributes to the rapid evaluation, redesign and reimplementing of each successive, improved, prototype.

In the GN&C RDL to date, a key feature of rapid development has been the use of advanced software development tools that tightly integrate design and analysis, documentation and automatic code generation.

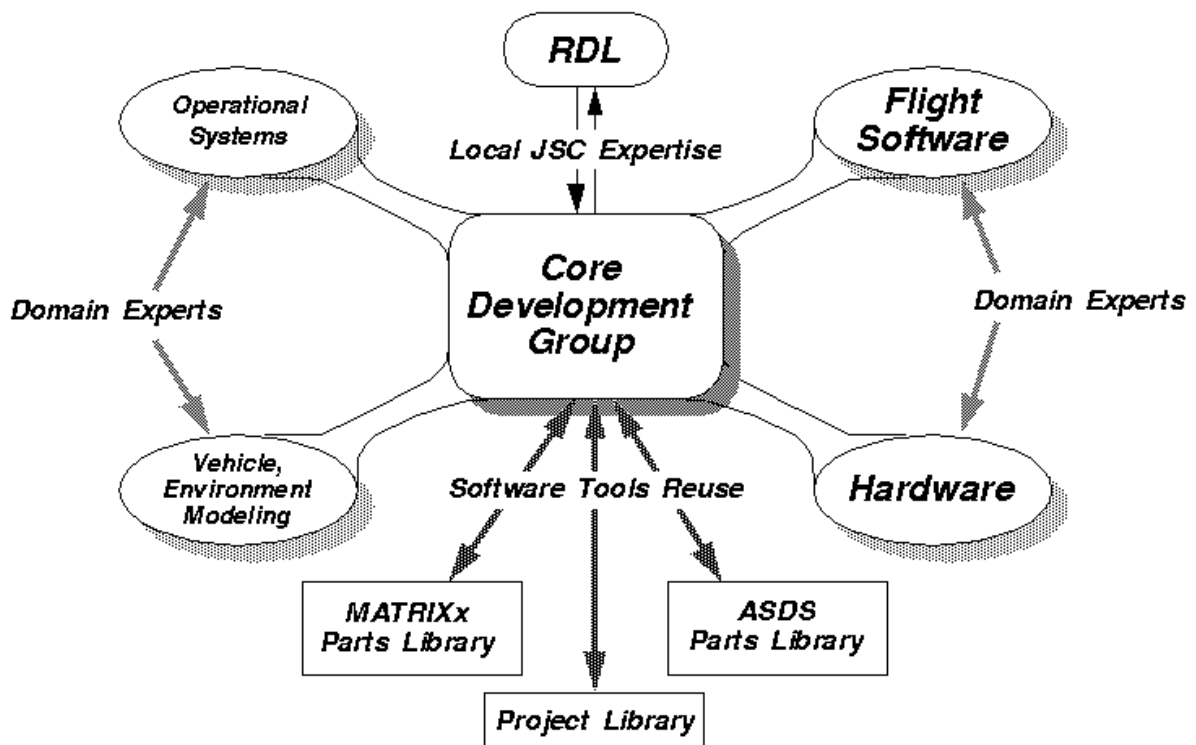


Figure 2. The Design Team Concept

3.0 The JSC GN&C Rapid Development Laboratory

The GN&C RDL is a JSC on-site resource dedicated to exploring, evaluating and applying new technologies and processes for flight software and simulation development, integration and testing. A variety of tools are available.

The use of advanced software development tools is central to the RDL approach to application development. An important tool currently in use in the GN&C RDL is Matrix_x, a COTS toolset sold by Integrated Systems Inc. (ISI), which provides an integrated environment in which to perform requirements analysis and application development, including design, code, test, and automated documentation, for the entire development cycle. This is a graphical software tool which allows the user to develop data flow block diagrams of the desired system using available primitives from a palette. Major capabilities include Xmath, SystemBuild, AutoCode, and DocumentIt, which are described briefly below.

Xmath - Matrix_x tool for design and analysis of control systems, simulations, and numerical calculations

- SystemBuild - Matrix_x graphical interface tool supporting system design from data flow block diagrams
- AutoCode - Matrix_x tool for translating SystemBuild block diagrams into Ada or C source code
- DocumentIt - Matrix_x tool for automated documentation and debugging support for FrameMaker, Interleaf, and ASCII environments

Other important tools for advanced software development in the GN&C RDL include MatLab, a technical computing environment for high performance numeric computation and visualization integrating numerical analysis, matrix computation, signal processing, and graphics; and ASDS (Advanced Simulation Development System), a generic trajectory and ancillary data GN&C/Propulsion simulation tool featuring extensive libraries of engineering models, utilities, and processes. MatLab is a COTS product marketed by MathWorks. ASDS was developed by McDonnell Douglas, the initial version under NASA funding.

A summary of the principal tools available in the GN&C RDL is found in tables 1 and 2 below.

Table 1. Software Tools in the RDL

Matrix _x	Rapid model development, control system design & analysis, code generation, simulation
MatLab	Engineering analysis tool
ASDS	Advanced Simulation Development System
SprocLab	Digital Signal Processing-based real-time development system
XILINK	Field Programmable Gate Array program development
ViewLogic	Graphic User Interface for Electronics CAD programs
ORCAD	Graphic User Interface for Electronics CAD programs
LabWindows	DOS-based device control, data acquisition and display
LabView	Window-based device control, data acquisition and display

Table 2. Hardware Tools in the RDL

3-Axis and 2-Axis rate tables
GPS (Global Positioning System) signal generator
Workstations: Sun, HP, PC, MacIntosh, SGI
Real-Time platforms: AC-100, DSPACE, STaTS, MATE, MDM, VME chassis with V _x Works
Mock-ups: Soyuz, Orbiter Aft Flight Deck

4.0 The Soyuz Vehicle Simulation

The goal of this project was to create a simulation environment for testing the Soyuz motion control system and related flight software with modifications to serve as a rescue vehicle for Space Station Freedom. This required simulation of vehicle characteristics and command responses for a variety of landing scenarios.

The existing flight software for the Soyuz descent module assumed the use of a large primary landing site in the former Soviet Union. At the time this project was undertaken, requirements called for the ability to land the vehicle under more accurate control and at other locations, including the United States and Australia. The project was terminated when these requirements were eliminated from the newly designed International Space Station.

The project was planned to include four major software deliveries, with increasing capability at each phase. Phase 0 included minimal functionality and primarily was a test of interfaces, system connectivity and communications. In phase 1, functionality of software modules was introduced, but was based on low fidelity 3DOF models using the characteristics of a familiar vehicle (Apollo). In Phase 2, simulation functions were rewritten to represent the Soyuz vehicle and flight software to the extent that the data was available. A physical mock-up of the descent module and some hardware performance data, such as thruster locations, mass properties and tank sizes, were available during phase 2 development, but knowledge of the flight software was still minimal. The Phase 2 simulation was validated by inviting Russian experts to "fly" the simulator and point out any obvious deficiencies or needed modifications. Phase 3, which was not implemented, was planned as a fully functional high-fidelity simulation.

This phased approach fit quite naturally with the project evolution, since it allowed progress to continue in the absence of complete detailed data. It also supported plans to use the project as a test for investigating the feasibility of evolutionary prototyping and the spiral development methodology to develop GN&C flight software.

This was also an initial test of the use of advanced software development tools to develop GN&C flight software.

The project ran from January 1993 through February 1994. Its results are documented in references 6, 7, 8, 9, 10 and 12 (section 9.0).

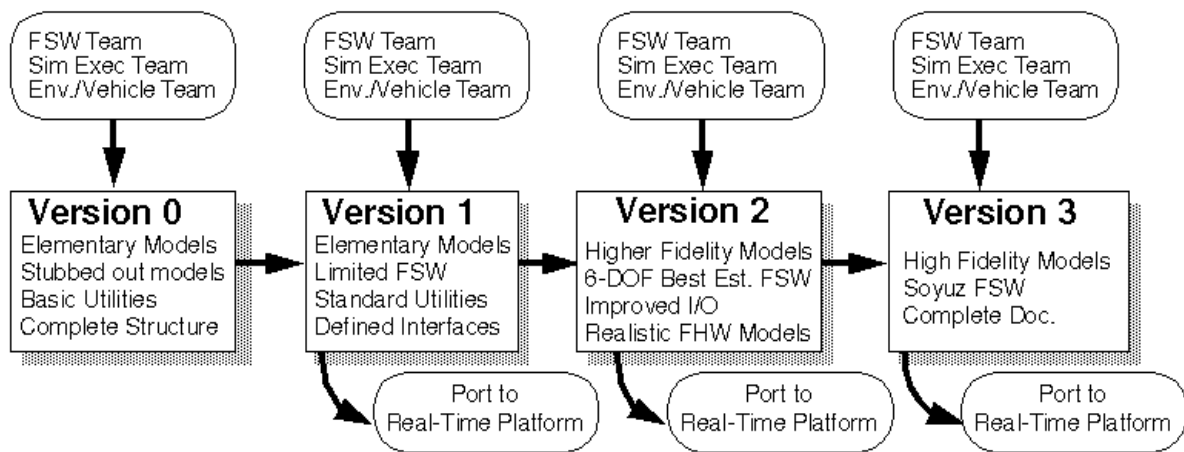


Figure 3. Soyuz Simulation Development Model

5.0 Rapid Software Development Results and Lessons Learned

The spiral development approach was effective for this project. By adopting the phased approach, with increasing levels of fidelity, it was possible to begin making progress on the project quickly, without having to wait for all data and information to be available. The early, low fidelity, simulations helped identify system structure and interface problems early on when they were easier to fix. The incremental deliveries created natural re-evaluation checkpoints for the project. And by delivering fully functional, though not high fidelity, interim prototypes a working simulation exists which could be shelved for potential future joint efforts with the Russians.

External dependencies (waiting on information or results from sources outside the project team) could still have strong effects on the project's progress, but in this development paradigm the team had more flexibility in responding to this type of problem. The team did note that success in this approach to system development depends heavily on putting together a small, well trained, diverse team. Customer involvement is crucial to evaluate prototypes between evolutionary steps. In fact, in this instance, the early simulations facilitated the discussion with the domain experts (including Russian engineers) to get performance data. Without an early prototype, the exchange of some important information would have been less successful. Sufficient time must be allocated for team meetings, meetings with users, demonstrations and evaluations of intermediate steps, and other communications-enhancing activities.

The application of the Matrix_x, SystemBuild and AutoCode tools were encouraging. The project team was able to document indications of significant productivity improvements for the coding phases of the project. (See Table 3 below.)

Table 3. Soyuz Simulation Phase 1 and 2 Metrics

	Phase 1	Phase 2
Number of Superblocks	55	371
Number of SLOC	4102	25045
Estimated Total Staff-Hours	1830	7720
SLOC per staff-day (assumes re-use of Phase 1 code)	18	22

The use of these advanced software development tools did bring its own set of problems. The tools were not mature, so discrepancies and idiosyncracies were frequently encountered. AutoCode-generated source was not always very efficient, and in some cases not even correctly generated. A side effect is that test sequences may have different results using SystemBuild and using the AutoCode source version of a system. Configuration management issues had not been effectively addressed in this environment. And the special needs of large scale projects were not well addressed, especially with respect to integration issues. The team investigated alternative COTS products, including MATLAB/SIMULINK (from MathWorks), VAPS (from Virtual Prototypes, Inc.), and tools in development at Honeywell, and concluded that Matrix_x was the most mature product set available. Since that time, the International Space Station (ISS) has used Matrix_x to develop close to 60% of its software, and this tool has improved significantly.

Considering the expected problems with the level of maturity of the tools at the time the Soyuz simulation project was implemented and the learning curve associated with first time application of the methodology and processes, the results of this project were considered to be very encouraging. This was a sufficient “proof of concept”, leading to establishment of a full-fledged Rapid Development Laboratory at JSC.

6.0 Topics for Further Study

With the Soyuz simulation project, the RDL team successfully applied the rapid development paradigm to a GN&C project. They applied the spiral development model, early end-to-end integration, evolving prototypes, and the core development team concept. They used estimated productivity improvements as a metric to achieve “proof of concept”.

Now we will expand on this work. The objective of this RTOP is to define and document generic reference processes for rapid development and integrated design and verification of

simulation and flight software. Furthermore, the RTOP will document the practical tailoring and application of these standard processes to project work in the GN&C RDL.

Some approaches to accomplish this include looking into other tools and approaches, looking for other examples of the concept, learning what kinds of development projects are amenable to the various tools and approaches, investigating the results that other groups have achieved, and expanding on what others have learned.

In the long term, there are many questions to consider (some of which may prove to be outside the scope of this RTOP). What other cutting edge approaches to system development (besides prototyping and auto-code generation, and spiral development) have been studied? Which of these methods have been tested? With what successes or failures, and under what circumstances? In what directions is technology advancement likely to lead us? What aspects of this technology have been proven to be useful in quality, dependable system development?

Configuration management issues and practices are concerns, as well as quality assurance and general software quality issues. There are issues to consider around industry and government standards, such as the Software Engineering Institute (SEI) CMM (Capability Maturity Model) and ISO9000. Developing useful metrics must be a goal.

In the Soyuz project lessons learned, there was considerable emphasis on the need to construct a small team of core experts to see the project through from start to finish. We will be interested in the critical characteristics of such a project team. For example, to what extent is the success of rapid software development projects dependent upon continuity or co-location of staff? What management structures are preferred? What skills and knowledge are indicators of success?

7.0 Early Thoughts and Special Concerns

Software development can be expensive and risky. If it were not, we would not, as a profession, be so driven to improve our methods. The classic goal is a methodology that produces better, faster, cheaper, or some combination of these. We need to add to this list, less risky and more flexible. Rapid prototyping development methodologies cannot make difficult problems easy. They cannot guarantee fast or cheap completion of software systems. This is not a panacea. Rather, many of today's system problems are so complex that advanced software development approaches are necessary if we are to solve them in a timely, useful and cost effective manner. Technology is advancing so quickly that a system that meets requirements frozen at some historical moment may be obsolete before it is delivered. All indications are that the complexity of the problems faced will continue to increase and exist in environments of constant change, and thus may require these new methodologies if we are ever to successfully solve them.

Some proven uses for software prototypes include:

- Mock-ups for selling or demonstrating an idea; these could be anywhere along a continuum from storyboarding to minimally functional systems.
- Determine and develop system requirements; this could also involve scenario development, to step a user through screens and actions, observe and record feedback, then modify the prototype and scenario, repeating the cycle until a solid understanding of the true system requirements is achieved. In this environment, the prototype does not itself evolve to the system, but serves first as a strawman and later as concrete visible documentation of system requirements.
- Analysis and what-if scenarios; in this environment, it is possible to try out several approaches and choose based on observations before making major commitments to system development.
- Evolutionary system development; this is the context of the Soyuz simulation development where, in the absence of full knowledge of the software requirements and hardware specifications, system development can evolve to increasing levels of complexity and accuracy as knowledge improves while serving to highlight system architecture problems and areas that need attention early in the development cycle.

When using prototyping for evolutionary system development, some advantages may be:

- The end product may be more responsive to current user needs since: 1) users had input into the process along the development cycle and 2) the product requirements evolve with the product, rather than being fixed at some historic point of requirements definition
- The end product may match user expectations more closely. Detailed written requirements specifications may be open to multiple interpretations. But by working together with evolutionary versions of a system, customers and developers have ample opportunity to converge on a common understanding of required system behavior.
- It may be possible to deliver the interim versions with some of the system functions working sufficiently to be useful. This allows users to begin getting some benefit from their investment sooner in the system life cycle, provides improved opportunities for generating user feedback in time to influence the system development, and provides an improved sense of progress.
- The evolutionary nature of the development cycle may result in a more robust and flexible system. It can continue to evolve and change with changes in the user environment.
- The process may control risk better than classic waterfall development methodologies, since customers/users have chances to evaluate the product as it is being developed. This gives better information to managers at project decision points and gives users chances to see and document problems, discrepancies, and deficiencies early in the system life cycle.
- Interface issues, frequently a source of last minute difficulties, are addressed early in the development cycle, often with the initial prototype.

When working in a rapid software development environment using evolutionary prototyping, the development environment must evolve as well. Since this is a leading-edge technology with rapidly evolving tools and techniques, this may imply an increased commitment to pur-

chasing new software tools, upgrading or replacing hardware, and investing in training classes for staff members.

There may be some additional difficulties integrating this approach into the government contracting environment or a business development cycle. We must look for ways to accurately estimate costs and completion dates while using an approach that depends on beginning a project before its endpoint is well defined. It will be especially important to devise metrics that track project progress and that are supportive of the spiral development environment while providing appropriate management feedback and accurate views of status.

Special attention must be paid to the question of scale. That is, are some approaches more effective in small or large scale projects? In the Soyuz simulation project, team members stressed the importance of a small tightly integrated team of experts. What implications does this have on large scale projects? A large project implies large coordination and integration efforts which exist regardless of the development methodology used. The methodologies may not scale up, or may require different or additional tools or approaches.

8.0 RTOP Plans and Schedule

Five major tasks are planned. These are:

1. Survey and review contemporary government and commercial sector rapid development processes.
2. Define and document draft guidelines for rapid software development technical, management and quality processes.
3. Define and document process performance metrics for rapid software development.
4. Correlate the draft guidelines and metrics for rapid software development processes to the Software Engineering Institute's Capability Maturity Model and to ISO 9000 requirements.
5. Demonstrate the application of the draft guidelines for rapid software development processes and metrics.

The work will be done as a joint project between NASA-JSC and the Jet Propulsion Laboratory (JPL). Tasks 1 and 2 are planned to be addressed in FY96, with deliverables for each. These include a Rapid Development Lexicon, a Literature Survey, and draft of proposed guidelines for rapid software development.

9.0 Related Documents

1. J.M. Ball & E.J. Riel, "Strategic Avionics Technology Program: Rapid Prototyping & Integrated Design Application Studies, Final Report"; NAS9-18877, February 1995.
2. J.M. Ball, D.C. Weed & E.J. Riel, "Strategic Avionics Technology Program: Rapid Prototyping & Integrated Design Application Studies, White Paper, Rapid Development Process"; NAS9-18877, February 1995.

3. B.W. Boehm, "A Spiral Model of Software Development and Enhancement", *Tutorial: Software Engineering Project Management* (R.H. Thayer, ed.), Computer Society Press of the IEEE, 1988, pp. 128-142.
4. B.W. Boehm, *Software Engineering Economics*, Prentice-Hall, 1981.
5. D. Pesek, "Rapid Development Lab Configuration Management Plan"; internal document, October 1995.
6. Rapid Prototyping Simulation Development Team, "Soyuz ACRV Simulation Development Progress Report, April-May 1993"; McDonnell Douglas TM-6.23.07-24; June 30, 1993.
7. Rapid Prototyping Simulation Development Team, "Soyuz ACRV Simulation Development Lessons Learned Report"; McDonnell Douglas TM-0009-01 enclosure 1; January 28, 1994.
8. Rapid Prototyping Simulation Development Team, "Soyuz ACRV Simulation Development Assurance and Test Report"; McDonnell Douglas TM-0009-01 enclosure 2; January 28, 1994
9. Rapid Prototyping Simulation Development Team, "Soyuz ACRV Simulation Development Configuration Management Plan"; McDonnell Douglas TM-0009-01 enclosure 3; January 28, 1994
10. Rapid Prototyping Simulation Development Team, "Soyuz ACRV Simulation Development Trip Report, Summary of Trip to MDA-Huntington Beach"; McDonnell Douglas TM-0009-01 enclosure 4; January 28, 1994
11. J. Uhde & D. Weed, "Library Reuse in a Rapid Development Environment"; *Proceedings of the AIAA Conference on Computing & Aerospace X*, March 28-30, 1995, San Antonio, TX; pp. 521-530.
12. J. Uhde-Lacovara, D. Weed, B. McCleary, & R. Wood, "The Rapid Development Process Applied to Soyuz Simulation Production", internal document, 1994

Rapid Development Lab Configuration Management Plan

Prepared by:
Douglas Pesek
Senior Engineer at McDonnell Douglas-Houston Division
(713)244-4517

Table of Contents

1.0	Introduction.....	5
1.1	Identification of Document.....	5
1.2	Scope of Document.....	5
1.3	Purpose and Objectives of Document	5
1.4	Document Status and Schedule.....	5
1.5	Document Organization and Roll-Out	6
2.0	Related Documentation.....	7
2.1	Parent Documents	7
2.2	Applicable Documents	7
2.3	Information Documents	7
3.0	Configuration Management Process Overview.....	9
3.1	Review Board.....	9
3.2	Software Engineering Staff.....	10
3.3	End User Community	10
4.0	Configuration Control	11
4.1	Configuration Control Responsibilities	11
4.1.1	Project Manager (PM)	11
4.1.2	Project Software Manager (PSM).....	11
4.1.3	Software Configuration Management (SCM) Group.....	11
4.1.4	Software Review Boards (CCB)	11
4.1.4.1	Current Status	12
4.1.4.2	Action Item Review	12
4.1.4.3	Change Request/Discrepancy Report Review	12
4.1.4.4	User Suggestions.....	12
4.1.4.5	Delivery Schedule	13
4.2	Product Delivery and Configuration Identification	13
4.2.1	Version Description	13
4.3	Configuration Change Control	13
4.3.1	Software Configuration Control.....	13
4.3.2	Incremental Integration and Testing Process.....	14
4.3.3	Version Upgrade Process.....	14
4.3.4	Documentation Process	14
4.3.4.1	Formulation Manual.....	14
4.3.4.2	User's Manual	14
4.3.4.3	Programmer's Manual	14
4.3.5	Mission Data Acquisition	14
4.4	Configuration Status Accounting	15

5.0	RDL Process Descriptions.....	17
5.1	Software Configuration Control Process	19
5.1.1	Overview of the change process	19
5.1.2	Description of files	20
5.1.3	Instructions for checking out Source Code	21
5.1.3.1	Checking out source code	21
5.1.3.2	Checking in source code	21
5.1.4	Process for adding or modifying entries in report forms	21
5.1.5	Process for generating status reports.....	22
5.1.5.1	RDL project bi-weekly test status report.....	22
5.1.5.2	RDL project test report.....	23
5.1.6	Allowable entries and definitions for Status and Priority.....	23
5.1.7	Process of archiving forms for a version release	24
5.2	Incremental Integration and Testing Process	25
5.2.1	Build Process Preparation	25
5.2.2	Normal Build Process: Code and Executable Upgrading.....	25
5.2.3	Normal Build Process: Verification.....	26
5.2.4	Abnormal Build Process: Corrective Action.....	26
5.3	Version Upgrade Process	27
5.3.1	Version Baseline.....	27
5.3.2	New Development Version Creation	27
5.4	Product Delivery Process.....	29
5.4.1	Version Description.....	29
5.5	Documentation Process.....	31
5.5.1	Formulation Manual	31
5.5.2	User's Manual.....	31
5.5.3	Programmer's Manual	32
5.6	Mission Data Acquisition Process.....	33
6.0	Acronyms and Abbreviations	35
7.0	Glossary.....	37
8.0	Appendices	41
8.1	GITF Processes	43
8.2	GITF Staffing	45
8.3	RDL Deliveries	47
8.3.1	Product delivery process.....	47

RDL Configuration Management Plan

1.0 Introduction

1.1 Identification of Document

The RDL Configuration Management Plan document contains a description of the software configuration management procedures and standards used for Rapid Development Lab (RDL) software products. This document conforms to NASA Software Documentation Standard, Software Engineering Program, NASA-STD-2100-91; specifically, NASA-DID-M600. This document was tailored after the MDA - Houston Division Software Configuration Management Guidelines document.

1.2 Scope of Document

This software configuration management plan was developed specifically for use on the RDL projects but may be adapted for use on other software projects.

1.3 Purpose and Objectives of Document

The purpose of the RDL Configuration Management Plan is to describe the software configuration management processes and standards that will be used to manage the RDL projects. These processes will define the framework for how changes to the products are controlled, how changes are managed, and how these changes are released. The RDL CCB will be responsible for maintaining and documenting all processes. The RDL configuration management processes are:

- Configuration Control
- Incremental Integration and Testing
- Version Upgrades
- Product Delivery
- Documentation
- Mission Data Acquisition

Included in 8.1 "GITF Processes" is a list of all the GITF processes with a brief description of each process.

1.4 Document Status and Schedule

This document is the initial release of the RDL Configuration Management Plan as a unique document. Revisions of this document will be released when significant changes are made to the processes contained herein. The appendices contain information that will be dynamic and will be updated as needed.

1.5 Document Organization and Roll-Out

This document is organized into 10 sections as follows:

- Section 1 identifies this document, defines the scope and purpose, presents status, and provides a brief description of each major section within this document.
- Section 2 lists references, related documents, and other applicable documents.
- Section 3 provides an overview of the configuration management process
- Section 4 describes the activities to be performed by the Configuration Control Board and associated members.
- Sections 5 includes a description of each RDL process.
- Sections 6 provides definitions of abbreviations and acronyms.
- Sections 7 is a glossary of terms.
- Sections 8 includes appendices that provide up-to-date information regarding configuration status and staffing, version releases, and other dynamic data.

2.0 Related Documentation

Other documents of importance are identified in this section.

2.1 Parent Documents

The following document is the parent from which this document's scope and content derive:

None.

2.2 Applicable Documents

The following documents are referenced herein and are directly applicable to this volume:

1. MDA-W, "Software Engineering Process Manual", Version 4.0, TBD.
2. NASA-JSC, "NASA Software Documentation Standard: Software Engineering Program", NASA-STD-2100-91, 29 July 1991.
3. MDA-HD, "Houston Division Software Configuration Management Guidelines", TBD

2.3 Information Documents

The following documents, although not directly applicable, amplify or clarify the information presented in this volume, and are not binding:

1. SEI, "Key Practices of the Capability Maturity Model", Version 1.1, CMU/SEI-93-TR-25, February 1993.

3.0 Configuration Management Process Overview

Software Configuration Management (SCM) activities include configuration identification, configuration control, and configuration status accounting.

Within the RDL project, many levels of configuration control are implemented to control various software products at various stages of development. Configuration control, at some level, is exercised over all final and interim products throughout the development cycle. The lowest levels of control are the responsibility of the engineers who are developing the product. At planned stages, the engineers turn the products over to the Software Quality Assurance (SQA) Group for verification of completeness and traceability, and to the SCM Group for control. When under the control of the SCM Group, products can be modified only after an evaluation of the system and project impacts of the proposed changes. The engineers, SQA Group, and the SCM Group exercise engineering control over products that are not part of a customer baseline and over products that are part of a customer baseline prior to establishment of that baseline. Once the customer approved baseline is established, procedures approved by the customer are followed to control changes to the baselined products.

A review board process is used to evaluate and disposition proposed changes to software products that have been put under the control of the SCM Group. The SCM Group, SQA Group, Project Manager (PM), Project Software Manager (PSM), review board, and Software Engineering Staff (SES) are responsible for fulfilling the requirements of the SCM function.

The configuration management process is concerned with the development of procedures and standards for managing an evolving software system. It is specifically concerned with change: how to control change, how to manage software systems which have been subject to change, and how to release these changed software systems to the users.

The product management, SCM, SES, SQA, and end users all participate in the flow of RDL products and information. Due to the size of the RDL projects, the SCM, SES, and SQA group functions are often performed by the same individual(s). Included in 8.2 "GITF Staffing" is a table of all members of the RDL team with a checklist indicating each member's role.

Change Requests (CR) and Discrepancy Reports (DR) are the official links between the product end users, the SES, and the specific project review boards. All requests must be reviewed and accepted by the appropriate product review board before being implemented into a released product. Implementation of any CR should be accompanied by documentation providing direct traceability for the changes made.

3.1 Review Board

The RDL project will have a product review board (hereinafter also referred to as the CCB - Configuration Control Board). The CCB, a forum created to direct the software development process, is populated by members of product management

(PM and PSM), the SCM Group, the SQA Group, the SES, and the end user community. The board meetings, which take place periodically, are the central platform for disseminating information, approving, prioritizing, and assigning software changes, and distributing products.

3.2 Software Engineering Staff

The Software Engineering Staff (hereinafter also referred to as developers) of RDL software are those designated to implement changes to the source code in response to inputs made by the product community through the CCB. These developers are responsible for attending the CCB meetings where CRs are prioritized and can be assigned to a developer. Once assigned a CR, the developer is responsible for the implementation of the requested change to the satisfaction of the author of the CR. If there are problems with the implementation of the requirements as requested by the CR author, the developer will contact the author and negotiate the difficulties. If, as a result, the substance of the CR changes, the new requirements should be reported to the CCB for approval and recording.

When changing code, the developer must follow defined procedures established for the specific software product. These procedures will assure traceability and recoverability of source code changes. In addition, the developer is required to produce a Change Action (CA) which records the changes made while implementing a CR and the affected source code modules.

While updating existing code or creating new code, developers are required to adhere to the standards and guidelines.

Developers also meet independently of the CCB to discuss schedules, assign CRs, and discuss ways to improve the quality and maintainability of the source code.

All developers for a software product must be recognized and approved by the product CCB. Only developers approved for work on a product may make changes to the baseline source code for that product.

3.3 End User Community

In addition to receiving the final software product, the product user community should take an active role in defining the requirements for the ongoing software development task. Change requests are drafted by users to define desired enhancements and corrections to the product. All change requests are presented to the CCB for approval and prioritization. The implementation of any change request must be approved by the author (e.g., signed off) before the request for the change can be considered closed.

The user community is encouraged to attend the CCB meetings and is invited to participate in the approval and prioritization process. Users can discuss any product related problems or concerns at the CCB. Specific items can be added to the CCB agenda by contacting the CCB Chair.

4.0 Configuration Control

4.1 Configuration Control Responsibilities

The purpose of this section is to identify and describe the activities to be performed by the configuration control staff.

The RDL project has individuals or groups designated to perform the duties and accept the responsibilities of the software project personnel identified in the following sections. Included in 8.2 "GITF Staffing" is a table of the engineers assigned to perform the functions described below.

4.1.1 Project Manager (PM)

The project manager has total business responsibility for the project and is ultimately responsible to the customer.

4.1.2 Project Software Manager (PSM)

The project software manager has total responsibility for all the software activities of the project. The PM deals with the PSM regarding all software commitments. The PSM has control of all software development resources or has a significant input into the use and control of shared development resources.

4.1.3 Software Configuration Management (SCM) Group

The SCM Group, all engineers performing specific SCM tasks, are members of the Engineering Staff who have been assigned SCM responsibility by the PSM. Most of the software engineers perform the SCM activity at lower levels of configuration control for the products they are developing. However, the SCM Group engineers provide engineering control for the products forwarded to them by the software developers.

4.1.4 Software Review Boards (CCB)

The Configuration Control Board (CCB) is a software review board populated by members of management (PM and PSM), the SCM Group, the SQA Group, the Engineering Staff, and the end user community of a specific product. The CCB meets periodically to exchange information, approve, prioritize, and assign software changes, and distribute products. No changes to products, versions of products, or distribution of products will be made without the knowledge and approval of one of the specific product's review boards.

The CCB is responsible for managing all RDL processes associated with product delivery and configuration control.

The CCB chair is responsible for establishing the CCB agenda and leading the CCB meeting.

The CCB secretary is responsible for reserving a location for the meeting, distribution of CCB meeting notices, maintaining a current distribution list, recording and distributing CCB meeting minutes. The distribution for the CCB includes the full community of users, developers, and management.

The CCB data base administrator is responsible for collecting and organizing all the change request materials required for the CCB meetings. This includes the collecting of paper and electronic CR forms, verifying that the forms meet the minimum submittal requirements, adding the new requests to the official data base, generating change request status reports, and editing all change forms to reflect the decisions of the CCB. A historical record of every CR for a given product is maintained by the CCB data base administrator.

The CCB meeting agenda should contain a detailed list of items to be discussed during the meeting. All attendees should be provided with handouts containing any information relevant to the agenda topics as well as a change request report that details the current status of all the active change requests in the official database.

The CCB agenda should cover, but not be limited to, the items defined in the following sections.

4.1.4.1 Current Status

The current status portion of the CCB meeting contains general announcements, software and/or documentation release briefing, answering questions raised at previous meetings, and other general business discussion. Included in the appendices is up-to-date information regarding configuration status including staffing, version releases, process owners and other dynamic data.

4.1.4.2 Action Item Review

The CCB will review the status of action items, especially those submitted since the previous meeting. Approval, prioritization, assignment, withdrawal, and closing of action items may take place at this time. A list of all action items is included in the appendices.

4.1.4.3 Change Request/Discrepancy Report Review

The CCB should review the status of change requests and discrepancy reports, especially those submitted since the previous meeting. Approval, prioritization, assignment, withdrawal, and closing of change requests and discrepancy reports may take place at this time. Lists of all change requests and discrepancy reports are included in the appendices.

4.1.4.4 User Suggestions

This portion of the CCB is an open floor discussion where anyone can discuss problems with the software, ask questions about processes, or make suggestions for improvement.

4.1.4.5 Delivery Schedule

The delivery schedule portion of the CCB meeting announces the current schedule for the next product deliveries. Any delays or adjustments to this schedule will be discussed at this time.

4.2 Product Delivery and Configuration Identification

The RDL product is identified by the product name and a version identifier. This combination of name and version identifies a unique configuration of the software product. Associated documentation will refer to this unique configuration using this configuration identification. Version identifiers indicate the chronological order of product releases. The procedures used for product delivery and configuration identification are described in detail in 5.4 “Product Delivery Process”.

4.2.1 Version Description

The Version Release Document provides a precise description of the particular version of the software being released. This description includes the requirements and design applicable to this version, and an exact description of the product contents. This document, which is published by the software engineers or an associated support group, accompanies every RDL software product delivery. For each new release, this document also provides information on the status of changes since previous releases. If a software product with an existing User's Guide is released without an updated User's Guide, the associated Version Description will contain detailed descriptions on how to use all new and enhanced features.

4.3 Configuration Change Control

The RDL project has established configuration control processes for all products, software, data, and documentation.

4.3.1 Software Configuration Control

The RDL project uses the Unix directory structure to support software configuration control and is described in detail in 5.1 “Software Configuration Control Process”. This process meets the following requirements:

1. All source code modules for each delivered product are recoverable.
2. Incremental changes to source code are identifiable, recoverable, and traceable to the initiating requirements.
3. Security/recoverability of the source code can not be compromised by persons outside the approved software engineering community.

All changes to RDL products are initiated by the submission of a CR or DR to the appropriate CCB. A CR/DR can be submitted by any user, manager, or developer to request a change, a problem fix, or an enhancement to the existing software products.

4.3.2 Incremental Integration and Testing Process

The RDL project has an established process for incrementally building new versions of the products. This process provides the mechanism for incrementally implementing CR's and DR's in a controlled fashion. Several builds will be performed between version releases. This process is described in detail in 5.2 "Incremental Integration and Testing Process".

4.3.3 Version Upgrade Process

The RDL project has an established process for upgrading and documenting new versions. This process is described in detail in 5.3 "Version Upgrade Process".

4.3.4 Documentation Process

The RDL has an established process for updating all RDL documentation. This process includes auto-generation of portions of this documentation. This process is described in detail in 5.5 "Documentation Process"

4.3.4.1 Formulation Manual

The Formulation Manual describes or derives the algorithm implemented in each of the models in software products. The terms model and package are used interchangeably in this document to mean a group of related functions used to perform a task. Assumptions and limitations of the formulations are discussed as completely as possible. References are included to provide the reader with additional background information and/or a more complete derivation of the algorithm. Updates to this document are only required when changes to the existing tools and algorithms warrant republication.

4.3.4.2 User's Manual

The User's Manual provides end users (rather than system operator or administrators) with instructions explaining how to utilize the software effectively. Republication of the User's Guide is appropriate for each product version release; however, user information for incremental releases (releases containing bug fixes and few, if any, enhancements) of software products can be supported by a combination of an existing User's Guide with a new Version Release Document for the given incremental release. Portions of this document are auto-generated using processors provided with all RDL products and tools.

4.3.4.3 Programmer's Manual

The Programmer's Manual describes the philosophy behind the program, explains its architecture, and specifies the procedures necessary to maintain and modify the program.

4.3.5 Mission Data Acquisition

The RDL project has an established process for acquiring mission data required to execute simulations. This process is described in detail in 5.6 "Mission Data Acquisition Process".

4.4 Configuration Status Accounting

The appendices of this document will provide up-to-date information regarding configuration status including staffing, version releases, process owners and other dynamic data for the GITF project.

5.0 RDL Process Descriptions

This chapter describes each process in detail and/or references additional materials which contain the necessary information. The processes described herein are:

- software configuration control
- incremental integration
- version upgrade
- product delivery
- documentation
- mission data acquisition

5.1 Software Configuration Control Process

This section provides a description of the RDL software configuration control process used to manage all changes to RDL projects. The CCB maintains a list of all Change Requests (CR), Discrepancy Reports (DR), Change Actions (CA), Action Items (AI), and a Wish List (WL) in the ~rdlog/ccb directory. This document describes the process for implementing changes to all project software, processors, data, documentation, and processes. The responsibilities of the CCB, developer, and end users are described.

Included in this document are:

- Overview of the change process
- Description of files
- Instructions for checking out source code
- Process for adding or modifying entries in report forms
- Process for generating status reports
- Instructions for users to submit new entries
- Instructions for filling out change action forms
- Allowable entries and definitions for Status and Priority
- Process of archiving forms for a version release

5.1.1 Overview of the change process

The following steps shall be performed to implement a change to RDL products.

1. User/Developer/CCB member (i.e. requestor) fills out the Master Problem Report Data Base.template describing the change in detail.
2. Requestor brings a copy of the form to a CCB meeting (see 5.1.6 "Allowable entries and definitions for Status and Priority").
3. The CCB decides to accept or decline the change request. If the request is accepted the CCB will:
 - a. Assign priority.
 - b. Determine which version the change will be applied to.
 - c. If the change is to be applied to the next release, then a developer will be assigned and the CR will be put into the CR form; otherwise the CR will be placed in the Wish List form and scheduled to be worked at a later time.
4. The developer will design and test the modifications required to close the change request:
 - a. The developer will research the change request to determine the extent of modification.
 - b. If the modification is complicated or if there are difficult design decisions, the developer may ask the CCB to provide direction.
 - c. The developer will open a Change Action form to record the modifications made.
 - d. The developer will use a directory copied from ~rdlog/version_X respectively in which to perform their code changes.

- e. The developer will check out (see 5.1.3 “Instructions for checking out Source Code”) all files that need to be modified. If a file is checked out already, the developer will bring this issue to the CCB and the CCB will assess the priority of the changes.
 - f. The developer will implement all changes, perform unit testing, schedule and have peer reviews performed, and determine integrated verification requirements.
 - g. The developer will bring the completed Change Action form to the CCB meeting for review.
5. The CCB will review the developers change action and decide to approve closing the request or to assign additional work. If the request is approved then:
 - a. The CCB will determine into which build the change will be implemented.
 - b. The developer will check in the modifications into the appropriate version (The CR/DR number will be supplied to the change using inline documentation or TEXT blocks).
 - c. The developer will supply the completed Change Action form to the Master Data Base directory.
 - d. The developer will supply one hard-copy of the final CR/DR and the Change Action forms to the CCB/builder to be filed in the project Closed Reports Notebook.
6. The CCB/builder will build the version with these changes and perform the integration testing specified in the Change Action form.
 - a. The CCB/builder will review with the developers results of those test cases which were identified in the Change Action form to have differences.
 - b. The CCB/builder will notify the requestor that the change has been implemented into the RDL product.
 - c. Finally, the CCB/builder will make a build sub-directory under the Change_Action_Forms directory and will move the Change Action forms which were included in the build to this sub-directory. The sub-directory name will be of the form *build.date* where *date* is the date of the build (i.e., *build.940725*).

5.1.2 Description of files

The following files contain the actual forms for all RDL CRs, DRs, Action Items, and Wish Lists:

- RDL Master Prob Rep Data Base

These forms contain the official list of CRs, DRs, Action Items, and Wish List Items and must be approved for inclusion by the project CCB. The Change_Request_Report is used to track all change requests for the current version under development. The Discrepancy_Report is used to track all discrepancy reports. The Action_Item_Report is used to track action items assigned by the project CCB. The Wish_List_Report is used to track all ideas for enhancements for future versions. These requests can be generated by the CCB, developers, or users.

The following files contain a status report on each of the lists:

- RDL Master Prob Rep Data Base

These files are generated from the report forms and are used by the CCB to track the progress made on each of the entries. The data contained in these table is a subset of the data contained in the actual forms.

A template for users to submit CR/Anomalies/DR/Action Items/Wish List is provided:

- RDL Master Problem Data Base.template

Developers for CR's and Anomalies/DR's are required to complete a Change Action (CA) form for each CR/Anomaly/DR. These forms are kept in the Change_Action_Forms directory.

5.1.3 Instructions for checking out Source Code

The Unix directory structure and permissions is used to manage the configuration control of all source code, data, and processors associated with RDL products. This section documents how the checkout is used for RDL products.

5.1.3.1 Checking out source code

After a change has been approved by the CCB, the developer will check out all files that need to be modified. The developer will only check out the files that he needs to make the enhancement/correction. A sign up sheet located in the ~rdlog directory for each version will be used to keep track which files are checked out of the delivered code. Changes made by any one else to a checked out code will not be accepted until the person signed up for the code has checked it back in. This will ensure that no one else can make modifications to that file or make changes to that file that would interfere with another developers work.

5.1.3.2 Checking in source code

When code is delta'ed back into configured directory for ~rdlog/version_X, the developer must submit a CR or DR number and the files that have changed. The files that are being replaced will be copied into a changed_code directory that resides at the level of the source code. The name of the file will remain the same as the original with the CR_# appended to it. This will ensure that at any time in development the earlier test cases can be created.

5.1.4 Process for adding or modifying entries in report forms

Only members of the project CCB can add to or modify the official forms. The process for adding or modifying an entry is the same for Action_Item_Report, Change_Request_Report, Discrepancy_Report, and Wish_List_Future_Versions.

RDL project anomalies can be entered in the scratchpad file in the "RDL Problem Reporting" folder. Any project test team member or their representative can open an anomaly report.

The steps for adding a new entry or modifying an entry in a report are:

1. The FileMaker Pro based version of the data base is on line on a Macintosh at Zone JSC B16 EG2, Server ADL MAC B16.
2. You will find what you need in the folder called "RDL Problem Reporting".
3. If you attempt to open the file "RDL Master Prob Rep Data Base", it will ask for a password. If you do not enter a password, you will only be able to browse the data base and print out selected records as any other user would. If you enter

the password at the prompt, you will have the ability to do anything you wish in terms of deletions, format changes, etc., so please be careful in this mode.

4. The other file titled "RDL Problem Reporting SCRATCHPAD", is a scratchpad on which users may make candidate entries to be turned into permanent entries at the discretion of the CCB. This is done by opening the file. It will ask for a password; just ignore the request and click OK. Then type command-N to call up a new record. Tab your way through the fields, answering the questions as you go.
5. When you have completed the record, you can stop or you may repeat the process to create more. When you are finally done with the new entries, you will want to bring them into the master data base.
6. Here's how to do that: First, you will have opened the master and supplied the password, since you are now writing, not merely browsing. Then, under the file menu, select import. You will tell the system that you want to import the records from the scratchpad file.
7. Once you have finished the import, type command-J to find all records, not just the ones you just pulled in. At this point, you will probably have to change the number that the program has assigned (at least if you want them to remain sequential). You will also notice that the scratchpad form is slightly different from the master data base in that several of the fields are suppressed. They are there, they are just hidden because the users don't need them. You can see that the tab key will cause you to cycle through these "phantom fields".
8. The system is set up to allow anyone to open the files (as read only), and it will limit access to the data files to one person at a time. Also in the folder you will find a Word and ASCII template of the report form.
9. Caution: If someone opens one of the files and just leaves it like that, this will effectively immobilize the whole system and won't let anyone else seize control of the specific file.

5.1.5 Process for generating status reports

5.1.5.1 RDL project bi-weekly test status report

Microsoft Word 5.0 test report summaries, as described below, will be requested from the test sponsors for two week periods.

- 1) Information will contain, as a minimum, the test case numbers, objective, success criteria, flight software used (hardware used if appropriate), and a technical synopsis of the results of each test case that was run.
- 2) For successful tests, the synopsis will include, as a minimum, how well the results agreed with the success criteria, which objectives were satisfied, and whether the test was completed.
- 3) For parts of tests in which anomalies in the flight article were found, the synopsis will include the RDL project DR number (or anomaly number if the DR does not exist) and a brief quantitative description of which success criteria was not met, which objectives were not satisfied, and the forward plan for flight article discrepancies resolution.
- 4) For parts of tests in which laboratory discrepancies prevented successful testing, the synopsis need only include the RDL project DR number (or the

anomaly number if the DR number does not exist) and the information in “1” above.

Test status reports from the various test summaries will be generated (condensed as appropriate) from the information given above and circulated to program management.

5.1.5.2 RDL project test report

Microsoft Word 5.0 test reports will be requested from the Testing Group at the completion of a series of tests. The Testing Group, in coordination with the Test Reporting Group, will determine what constitutes a “series” of tests. The Test Reporting Group, in coordination with the Testing Group, will develop a test report standard. The standard will be concurred upon by the ISS Program representative and approved by the Program Management Group prior to use. The Test Reporting Group will generate a schedule for test reports based on the information provided by the Testing Group.

The Test Reporting Group will review test report schedules with the Testing Group and arrive at report delivery dates that reflect ISS program needs and the RDL project schedule capabilities.

Test reports will be reviewed for format and completeness by the Test Reporting Group and distributed once release approval is obtained from the Program Management Group. A library of released test reports will be maintained by the Test Reporting Group in Microsoft Word 5.0 electronic form.

5.1.6 Allowable entries and definitions for Status and Priority

Table 1: Allowable entries and definitions for the status block

Entry	Definition
Declined	Proposed changed was withdrawn
Accepted	Approved for implementation, no developer assigned, priority not set
On-Going	Developer assigned, being worked
On-Hold	Developer assigned, not being worked
Pending	Code placed into a build but CR/DR left open because of missing CA item(s)
Closed	Item approved for closing by the CCB

Table 2: Allowable entries and definitions for the priority block

Entry	Definition
High	Will be worked as soon as possible
Medium	Will be worked as resources become available
Low	Will be worked in slack time

5.1.7 Process of archiving forms for a version release

All AI, CR, DR, and CA forms must be archived when a new version of RDL is released.

The steps for archiving these forms are:

1. `cd ~"rdlog"/ccb`
2. Create a new directory for storing the archived forms, naming the directory: `Version_X_Forms` (where X is the version number).
3. Copy all files that are in the `Report_Book` (including the `Report_Book` file) from `~"rdlog"/ccb/Report_Book` to the `Version_X_Forms` directory.
4. Create a `Change_Action_Forms` directory in the `Version_X_Forms` directory and move all of the CA forms that were implemented into the new version from `~"rdlog"/ccb/Change_Action_Forms` to this new directory.
5. Make sure that all CR and DR that were implemented into the new version have been closed and that a CA exists for each CR and DR.

This completes the archiving process. Now the completed CR, DR and AI forms must be removed from the `~"rdlog"/ccb` table to prepare for the next version.

The steps for removing these forms are:

1. For each of the tables (`Action_Item_Report`, `Change_Request_Report`, `Discrepancy_Report`) perform the following steps:
 - a. Open the table file.
 - b. Remove the forms that have been closed from the table by deleting the pages containing the closed form.
 - c. On the first body page of the form change the paragraph type to be `Page1`.
 - d. Using the paragraph designer, modify `Page` definition for numbering.
2. The `Wish_List_Future_Versions` should be reviewed by the CCB to determine which of the items should be moved to the CR list.
3. Create new status reports for each file (process described in this document).

5.2 Incremental Integration and Testing Process

RDL projects have an established process for integrating and testing incremental updates to new versions of the product. This process, hereafter referred to as the **build** and **build process**, provides the mechanism for incrementally implementing CRs and applying DR fixes in a controlled fashion. Several builds will be performed between version releases. The person performing the build process will be referred to as the **version builder** or simply the **builder**.

5.2.1 Build Process Preparation

The CCB first decides that a build update will be performed and which CRs and DRs will be included in that build. The builder must prepare by identifying all the source code and data files to be used in this particular build.

1. A build date must be selected that is indicative of the date the build will be initiated using a 6 character format (2 characters each) for year, month and date, such as 940920 for September 20, 1994. This mnemonic (940920) will be referred to as the **build_date**.
 - a. The **build_date** in the code file `~"rdlog"/version_x/Readme_build` is updated to the new date.
2. Review the Change Action Form (CA) for each CR and DR to become familiar with what is expected with this build. This is important especially when test case results are expected to change and the builder must use this information to assist in the build verification process.
3. The builder then updates the build report's, **Readme_build**, initial section denoting what CRs and DRs expect to have test case differences. This information should be available in the respective CA based on the details that were included in the CA supplied by the developer. This information is then used to assist in the build verification process discussed later.

5.2.2 Normal Build Process: Code and Executable Upgrading

The build process can now be performed. There is currently no automated means of configuration management in MatrixX, so documentation is crucial. The following steps are used to perform the actual build process.

1. The builder, having checked out all the files he needs, modifies the code accordingly.
2. When modifying a SuperBlock, document the changes using a Text Block. The SuperBlock's comment field is usually used for official program unit documentation. Things to document are: the name, date, version, and other important identification data. If appropriate, document the rationale for any changes that may need it.
3. Manual configuration management introduces some specific cautions.
 - a. Always save SuperBlock changes in ASCII format.
 - b. Be careful not to document changes that have not been completed. We want to avoid the situation where a builder thinks a change has been made when it really hasn't.
 - c. On the same line, remember to document all changes that have been completed.

4. Warning: As a means of determining the changes between simulation files, the unix command 'diff', is not sufficient. That is why detailed documentation of changes is so important.
5. After all changes have been made, documentation is complete, and the simulation is autocoded, the builder needs to compile and link the source code.

5.2.3 Normal Build Process: Verification

When there are no errors in the compilation and linking of the source code, and all test cases have been executed the following steps must be completed.

1. The test case verification process by reviewing the ".diff" output file of each test case executed. Based on the review of the CA for each CR and DR discussed earlier and now documented in the **build report**, the builder must have a working knowledge of what test cases should have differences and what degree of difference expected.
2. Normally, the actual differences of each test case should match the expected differences for that test case obtained through the review of the applicable CAs. The builder should also review the computer timing usage for each test case to determine if any test case(s) experienced a significant computer time usage change. After reviewing all test case results, the builder should make any necessary comments/remarks in the build report for future reference. Trajectory plots of selected trajectory parameters are then generated for those test cases having differences. A copy of these plots along with a copy of the final build report is then filed for permanent retention.

5.2.4 Abnormal Build Process: Corrective Action

The build process which integrates the incremental code/data upgrades does not always perform normally. Even though each individual developer has gone through an extensive verification process for each CR and DR implementation, the final build process integrates all changes at one time and possible conflicts between CR and DR implementations can arise. Other problems can occur, whenever the build process explained above does not finished successfully, the builder has to take the necessary steps to identify the problem(s) and to make the necessary correction(s).

1. If minor code corrections are required. The builder should make the corrections in the actual source code.
2. Another possible build problem experienced is one with data problems within the test cases themselves. The builder should correct the data files having problems.

If major code corrections are required, then the builder may have to make the code corrections within the source code directory with the assistance of the developer.

5.3 Version Upgrade Process

RDL projects have an established process for upgrading and documenting new versions. This section describes the process of baselining the current version of the project software (excluding validation testing) and creating a new version of the software for continued development.

5.3.1 Version Baseline

Once the CCB has decided that the current project version meets the needs for a particular milestone, that version must be baselined and a new development version made available to the development community.

The deliverable (current) version of the software is prepared for baselining by locking the software configuration files from the development community. This baselined version is used for validation and testing, and when this is completed, is ready for distribution to all customers.

5.3.2 New Development Version Creation

Once the current version of the software has been baselined, a new development version is required for the processing of new Change Request and Discrepancy Report corrections and modifications.

1. Write a CR that states to create a new version of project software. This CR should be number **Y.1** where **Y** is the new version number.
2. In directory `~"rdlog"`, create a new directory structure which begins with contents that are identical to the development version (baseline).
3. Edit the new version root (`~"rdlog"/version_Y`) Makefiles
Update items that might have changed in the Makefiles
4. Present results of the creation of the new version to the CCB. Review with the CCB the following pieces of information:
 - a. any problems found during the new version creation process.
 - b. mission test cases that need to be deleted.
 - c. go/no-go recommendation for the first build in the new version.
5. Perform the version build (see **5.2 Incremental Integration and Testing Process**).
6. Verify the build (see **5.2 Incremental Integration and Testing Process**). There should be no differences between the new version (`~"rdlog"/version_Y`) results when compared to the results from the previous, baselined version (`~"rdlog"/version_X`) of RDL.
7. Present results to the CCB.
8. Unlock the permissions in the directories in the new version.
9. Announce the existence and readiness of the new version to the developer community.

5.4 Product Delivery Process

RDL products are identified by the product name and a version identifier. This combination of name and version distinguishes a unique configuration of the software product. Associated documentation will refer to this unique configuration using this configuration identification. Version identifiers indicate the chronological order of product releases.

5.4.1 Version Description

The Version Release Document provides a precise description of the particular version of the software being released. This description includes the requirements and design applicable to this version, and an exact description of the product contents. This document, which is published by the software engineers or an associated support group, accompanies every RDL software product delivery. For each new release, this document also provides information on the status of changes since previous releases. If a software product with an existing User's Guide is released without an updated User's Guide, the associated Version Description will contain detailed descriptions on how to use all new and enhanced features.

The product delivery process is described in detail in 8.3 "RDL Deliveries".

5.5 Documentation Process

RDL projects have an established process for updating all associated documentation. This process defines what documents need to be updated as coding changes are implemented and what activities must be performed for a version delivery.

When an RDL product coding change is made, the code developer is required to identify all affected documents and update the appropriate text sections before the coding change is considered closed. The updated sections are reviewed by a co-worker for comprehensibility and completeness. The type of coding change determines which documents need to be updated, and this is discussed in subsequent sections.

The sections of a document may evolve as coding changes are implemented for the next version release. All documents are then re-created at the version release to incorporate all the changes. This includes assembling the sections of the document into a book, updating section numbers and cross-references, creating the table of contents, and printing the document. For the User's Manual, this also includes input, output, and list tables directly from RDL product code files.

5.5.1 Formulation Manual

The Formulation Manual describes or derives the algorithm implemented in each of the models in the RDL products. The term model means a group of related functions used to perform a task. Assumptions and limitations of the formulations are discussed as completely as possible. References are included to provide the reader with additional background information and/or a more complete derivation of the algorithm.

This manual describes:

- The executive package used to control events and integration
- The various coordinate systems and reference times
- The computation of the time derivative of the vehicle state
- The environment models: atmosphere, gravity, and winds
- The vehicle models: aerodynamics, mass properties, propulsion, sensors, and slosh
- Various utility functions including interpolation and spline fit routines
- The hardware configuration necessary to run the simulation

Updates to the Formulation Manual are required when a change is made to the engineering or mathematical algorithms used by a model or a hardware configuration change. For example, new aerodynamic equations or additional inputs would require a change to the aerodynamics section of the Manual, but a change in data values without a change in format or use would not require a change to the Manual. The developer would document the actual equations used, explaining the terms, and include explanatory figures, if necessary.

5.5.2 User's Manual

The User's Manual provides end users (rather than developers or administrators) with instructions explaining how to execute the software effectively. The Manual

provides information needed to enter appropriate data, assemble a simulation, obtain output data, and configure necessary hardware.

The Manual is arranged by model packages, and each model package usually includes an overview section, sample input, and an error control section. If applicable, tables of inputs, and outputs follow for each package.

Updates to the User's Manual are required when a change is made that affects examples, error controls, inputs, or outputs. The developer would make appropriate changes to any text (such as examples and error controls) when the coding change is implemented but would not update any of the tables.

5.5.3 Programmer's Manual

The Programmer's Manual describes the philosophy behind the RDL program, explains its architecture, and specifies the procedures necessary to maintain and modify the program. It documents the development requirements (including operating system and language requirements), documents the design concepts and coding conventions used for program development, discusses the functional design requirements for the program's components, and explains the procedures and tools used to maintain and modify the program.

Updates to the Manual are made when the program's basic philosophy or architecture are changed. For example, if the method for adding new derivatives changed or if a new function was required for each model package, the developer would document the new requirement in the Manual.

5.6 Mission Data Acquisition Process

The Mission Data Acquisition Process (MDAP) is an important part of the RDL maintenance. This process builds the flight-specific databases needed to support flight verification and mission analysis using RDL products

Flight-specific data can come from a number of sources and in a variety of forms. The purpose of the MDAP is to transform these different types of data into RDL product input data with minimal manual intervention.

The steps in the MDAP are described extensively in the MDAP document (~"rdlog"/ccb/MDAP/BOOK) due to their complexity. Basically, there are three parts to the MDAP:

1. Gathering the input data
2. Executing the processors to generate RDL product data files
3. Verifying the results

Once all the input data have been obtained, a number of processors and scripts are used to convert these data to RDL product mission-specific data files. Types of output data files are I-loads, initialization file, mass properties, and propulsion. These mission-specific data files are then added to an RDL master data file template to create mission-specific test case master data files.

The verification of the MDAP results is performed by running a standard set of mission-specific test cases and examining the results. Again, scripts are used to automate the creation, execution, and plotting of the test cases. The user then examines the test case output (text files and plot files) to ensure that the mission-specific data got in correctly and that the test case ran correctly.

6.0 Acronyms and Abbreviations

CA	Change Action
CCB	Configuration Control Board
CI	Configuration Item
CMM	Capability Maturity Model
CR	Change Request
CRM	Change Request Manager
CSCI	Computer Software Configuration Item
DID	Data Item Descriptions
DR	Discrepancy Report
ECP	Engineering Change Proposal
GITF	Guidance, Navigation, & Control Test Facility
HD	Houston Division
ISS	International Space Station
JSC	Johnson Space Center
MDA-W	McDonnell Douglas Aerospace - West
NASA	National Aeronautics and Space Administration
PM	Project Manager
PSM	Project Software Manager
RDL	Rapid Development Lab
RDLOG	Project Name
SA	System Administration
SCCS	Source Code Configuration System
SCM	Software Configuration Management
SDR	Software Discrepancy Report
SEI	Software Engineering Institute
SEPG	Software Engineering Process Group
SES	Software Engineering Staff
SMAP	Software Management and Assurance Program
SQA	Software Quality Assurance
STS	Space Transportation System
TM	Transmittal Memo
VDD	Verification Description Document

7.0 Glossary

Change Action (CA)

A form written by developers to describe the implementation of a change request.

Configuration Control Board (CCB)

A forum created to direct the software development process and populated by the user community, management, and the developers of each specific product. The board meetings, which meet periodically, are the central platform for disseminating information, approving, prioritizing, and assigning software changes, and distributing products.

Configuration Item (CI)

An aggregation of hardware, software, or both, designated for configuration management and treated as a single entity in the configuration management process.

Change Request (CR)

A form written by users or developers to describe either a discrepancy between the software and a requirement, or a software enhancement to support a new requirement.

Change Request Manager (CRM)

An on-line, electronic application developed by the Robotic Systems Project which manages change requests and developer change action reports.

Computer Software Configuration Item (CSCI)

A software specification item whose function and performance parameters must be defined and controlled to achieve the overall end use function and performance.

developer's forum

A product developer's meeting attended by the lead developer, the CCB chair, and at least half of the developers. This forum is able to act with the power of the CCB for actions of high priority that have critical time constraints.

discrepancy (NRCA) report

A report defined by the NASA Software Documentation Standard used to state a discrepancy to a product or product specification.

Engineering Change Proposal (ECP)

A report defined by the NASA Software Documentation Standard used to state a suggested change to a product. For HD purposes, this is equivalent to a CR.

formally controlled software

Software, deliverable to an external or internal customer, or any software significantly relating to acceptability of deliverable hardware or software items.

guidebook

A handbook of information; within the MDA-W software process documentation scheme, a manual with non-directive information, supporting compliance with directive information contained in company manuals and procedures.

NASA-STD-2100-91

This document describes the standard method of documenting software at NASA. It has been adopted by HD as a standard set of software documentation guidelines and templates.

organization

A unit within a company or other entity (e.g., government agency or branch of service) within which many projects are managed as a whole. All projects within an organization share a common top-level

manager and common policies. The Houston Division is considered an organization within the SEPM framework.

periodic

(1) Having periods or repeated cycles; (2) Occurring or appearing at regular intervals; (3) Taking place now and then: intermittent [Note: The time interval will meet the needs of the organization and may be lengthy.]

Project Manager (PM)

The individual having total business responsibility for a project and being ultimately responsible to the customer. In most instances, this designation will be given to the appropriate E level manager within HD.

Project Software Manager (PSM)

The individual having total responsibility for all the software activities of a project. The PM deals with the PSM regarding all software commitments. The PSM has control of all software development resources or has a significant input into the use and control of shared development resources. In most instances, this designation will be given to the appropriate F level manager within HD.

software baseline library

The contents of a repository for storing CIs and associated records.

Software Configuration Management (SCM) group

The SCM Group, all engineers performing specific SCM tasks, are typically part of the SES who have been assigned SCM responsibility by the PSM. Most of the software engineers perform the SCM activity of lower levels of configuration control for the products they are developing; however, the SCM Group engineers provide engineering control for the products forwarded to them by the software developers.

Software Engineering Process Group (SEPG)

A group facilitating definition, maintenance, and improvement of the eternizations software process. The HD SEPG was formerly known as the Software Process Improvement Team (SPIT).

Software Engineering Staff (SES)

All software technical personnel, other than the PSM, performing software development activities.

software product

The complete set, or any of the individual items of the set, of computer programs, procedures, and associated documentation and data designated for delivery to a customer or end user.

software review board

A group of technical and management representatives and technical experts whose function is to evaluate, approve or disapprove, and coordinate proposed changes to configuration items, has the following specific configuration control responsibilities.

software work product

Any artifact created as part of defining, maintaining, or following a software process, including process descriptions, plans, procedures, computer programs, and associated documentation, which may or may not be intended for delivery to a customer or end user.

system

A collection of components organized to accomplish a specific function or set of functions.

Unix

A computer operating system conceived in 1969 at AT&T's Bell Labs that offers, among other features, file and record locking, user security, multi-tasking, electronic mail, networking, and remote file sharing.

Version Description Document (VDD)

A document describing the version of software to be released and changes from the previous version, if applicable. The Houston Division standard template for this document is NASA-DID-P500 (found in NASA-STD-2100-91).

walkthrough

An ad hoc team activity conducted by development peers to ensure a software product under development satisfies applicable requirements and design objectives.

workstation

A table-top sized mini-computer intended primarily for a single user. Equipped with a CPU, monitor, keyboard, and memory (both static and dynamic), and can also support tape drive/s and hard drive/s. Workstations are usually part of a network.

8.0 Appendices

8.1 GITF Processes

This section describes the processes used for configuration management for the GITF project and designates the process owner. The process owner is responsible for maintaining the process and associated documentation.

TABLE 8-1 GITF Processes

Process Name	Owner	Description/Status of Documentation
Configuration Control	Karen Frank	Defines the process for modifying GITF code, processes, databases, documentation, and processors and defines the roles and responsibilities of CCB members, developers and users
Incremental Integration and Testing	Bruce Shulz	Defines the process for incrementally implementing modifications (builds) leading up to a new version.
Version Upgrade	John Craft	Defines the process for upgrading GITF products to a new version including verification requirements and documentation requirements.
Product Delivery	Frank Weaver	<p>Defines the procedures followed to deliver final GITF test reports to customers.</p> <p>Inputs to this process includes all source code identifiers to create the required executables, as well as certification test case-numbers to test the executables and all shell script numbers required to reconfigure the simulation.</p> <p>Additionally, test sponsors submit written reports on the test objectives and results. Reference is made to the information above. These reports, once approved, become the GITF formal "product delivery"</p>
Documentation	Jenny Wagenknecht	Defines the process for maintaining all GITF documentation including the process of extracting input/output definitions from the source code.
Mission Data Acquisition	Teming Tse	Defines the process for obtaining mission-specific data including I-loads for all flight modes, mission-specific data files and master data files for each flight mode.

8.2 GITF Staffing

This section documents the management and engineering staff for the GITF project. Due to the staffing level of the GITF project, engineers supporting this project are assign multiple duties and responsibilities.

TABLE 8-1 GITF Staffing

Engineer	Project Manager (PM)	Project Software Manager (PSM)	Software Configuration Management Group (SCM)	Configuration Control Board (CCB)	Software Engineering Staff (SES)	Software Quality Assurance (SQA)
NASA Support						
						✓
						✓
						✓
						✓
						✓
						✓
						✓
						✓
Contractor Support						

8.3 RDL Deliveries

This section documents the delivery process for all RDL products and to whom products have been delivered. The table below shows the RDL version deliveries that have been made to date.

Table 3: RDL Deliveries

Config	Description	Delivery Date	Version

8.3.1 Product delivery process

The steps needed for RDL product delivery are:

- All CRs and DRs need to be signed off on by the author of the report
- Customer must concur with the changes implemented
- All documentation must be completed

LIBRARY REUSE IN A RAPID DEVELOPMENT ENVIRONMENT

Jo Uhde Ph.D.
NASA Johnson Space Center
Houston, Texas 77058

Daniel Weed, Robert Gottlieb Ph.D., Douglas Neal
McDonnell Douglas Aerospace - Space Systems
13100 Space Center Boulevard.
Houston, Texas 77059

ABSTRACT

The Aeroscience and Flight Mechanics Division (AFMD) at the National Aeronautics and Space Administration-Johnson Space Center (NASA-JSC) established a Rapid Development Laboratory (RDL) to investigate and improve new "rapid development" software production processes and refine the use of commercial, off-the-shelf (COTS) tools. These tools and processes take an avionics design project from initial inception through high fidelity, real-time, hardware-in-the-loop (HIL) testing.

One central theme of a rapid development process is the use and integration of a variety of COTS tools such as the MATRIX_X[®] product family of Integrated Systems Inc. (ISI), Santa Clara, CA, and the Advanced Simulation Development System (ASDS), developed by NASA and McDonnell Douglas Aerospace in Houston, TX (MDA-HD).

Reuse of MATRIX_X[®] and ASDS models is crucial to the rapid development process. Unlike ASDS, which is expressly designed to facilitate software reuse, MATRIX_X[®] poses some unique problems not associated with reuse of source code models.

This paper¹ discusses the RDL MATRIX_X[®] libraries, as well as the techniques for managing and documenting these libraries. This paper also shows the methods used for building simulations with the ASDS libraries, and provides metrics to illustrate the amount of reuse for five complete simulations. Combining ASDS libraries with MATRIX_X[®] libraries is discussed.

INTRODUCTION

In an effort to evaluate the rapid development process and tools, AFMD created the Rapid Development Laboratory. The RDL is a JSC on-site resource dedicated to exploring and evaluating new technologies and processes for flight software (FSW) and simulation development. [Bordano, 1993]

The laboratory is a teaming effort between personnel from NASA-JSC, Lockheed Engineering Services Corporation (LESC) and MDA-HD. A cooperative agreement allowed engineers at JSC to access a similar facility at the McDonnell Douglas Aerospace site in Huntington Beach, CA (MDA-HB), where the flight software and simulations for the Delta Clipper (DC-X) were developed.

The RDL is fine tuning a spiral, or incremental, development process, wherein entire software and hardware systems are integrated early in the development cycle. The system level of fidelity increases through a series of milestones toward the desired final state. This process has been demonstrated to substantially increase productivity over the life of a project compared to more traditional, "waterfall" approaches.

The RDL team uses the MATRIX_X[®] product family available from Integrated Systems Inc. (ISI), Santa Clara, CA, to build entire simulations in block diagram form. The Advanced Simulation Development System is used separately or in conjunction with MATRIX_X[®] depending on the simulation requirements.

Software Reuse and the Rapid Development Process

Simulation building in the past has been laborious and expensive, both in development and maintenance. Simulations were traditionally built to satisfy a unique set of requirements without regard to reuse of any kind. Reuse of these tools was typically an afterthought, leading to higher development and maintenance costs. An improved software development approach would combine extensive software

1. Copyright 1995 by the American Institute of Aeronautics and Astronautics, Inc. No copyright is asserted in the United States under Title 17, U.S. Code. The U.S. Government has a royalty-free license to exercise all rights under the copyright claimed herein for government purposes. All other rights are reserved by the copyright owner.

reuse with spiral development techniques and integrated software engineering environment tools (SEE)

The spiral development approach produces prototype real-time simulations and code concurrently with the requirements development. Once the first working prototype of the FSW is developed, real-time hardware-in-the-loop (HIL) testing is initiated with representative flight hardware.

This process allows incompatibilities in the software design, implementation, or in hardware selection to be discovered early in the development. The cycle of concurrent requirements and software development, and HIL testing is repeated until acceptable software is produced. The spiral development approach is one in which developers “build a little and test a little”.

Crucial to the rapid development process, and one of the main characteristics that makes it “rapid” is the reuse of existing software, test cases, and documentation. ASDS was specifically designed with reuse in mind. However, the full suite of MATRIX_X[®] tools are primarily used as an integrated software engineering environment. Such an integrated toolset, in which one development environment is used from initial requirements specification through final real-time validation, is also important to rapid development projects. But it is not as well suited to software reuse as more

traditional programming languages.

Overview of MATRIX_X[®]

MATRIX_X[®]/SystemBuild[™] is a graphical software tool which allows the user to develop data flow block diagrams of the desired system using primitives available from a palette. These elementary blocks are organized in groups called “SuperBlocks” which become procedures or subtasks. These SuperBlocks contain other hierarchically nested SuperBlocks, which may be duplicated and shared among other SuperBlocks.

Once the software data flow diagrams are built and linked together, they are interactively tested in a non real-time environment. Time and frequency domain analysis are also performed interactively. An example of a block diagram is given in Figure 1. This diagram illustrates the use of several elementary blocks and imbedded SuperBlocks, which in this case are library utilities.

Real-time code is automatically generated using the AutoCode[™] tool to produce source code from the block diagram representation in C or Ada. Legacy code may be imported into the model via User Code Blocks. The source code is compiled and run on the AC-100[™] real-time computer to verify real-time and HIL performance.

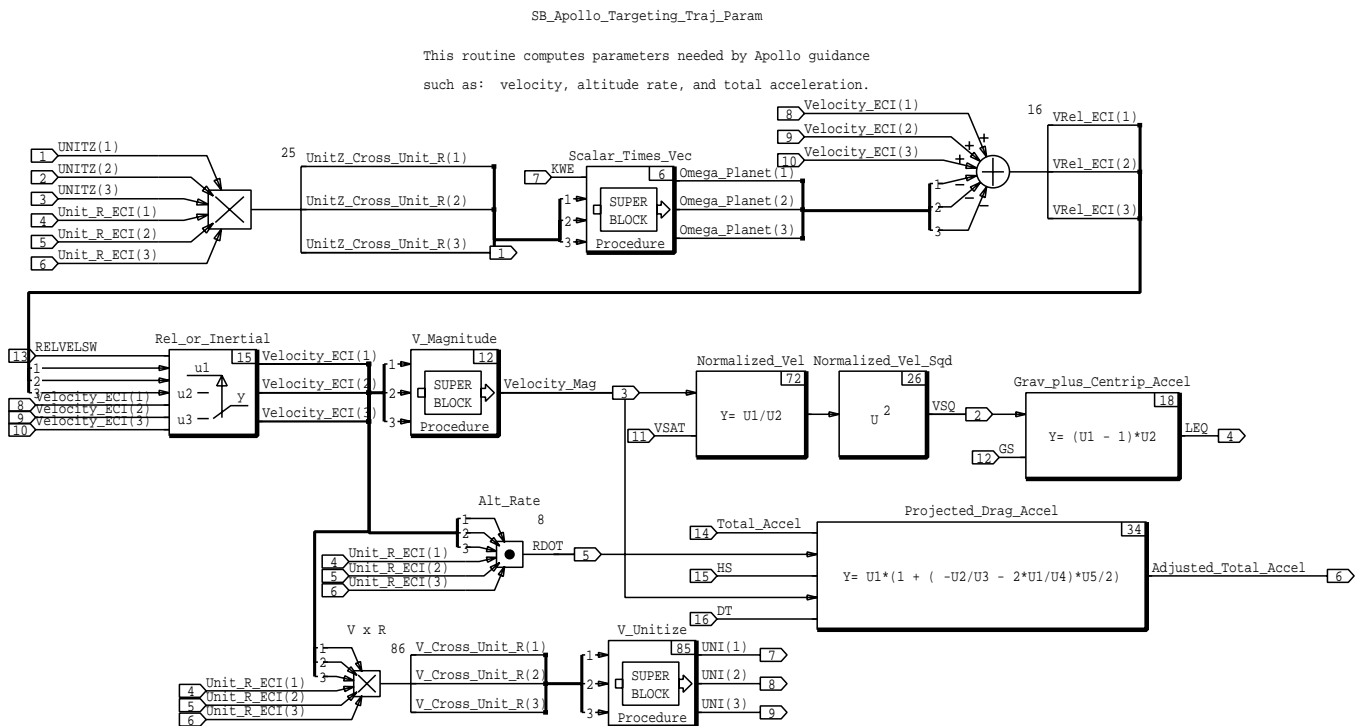


Figure 1 Example MATRIXx Block Diagram

Overview of ASDS

The Advanced Simulation Development System (ASDS) is a unique system in which a library of software parts has been successfully reused, at all levels of complexity, in the assembly of high-fidelity, large-scale simulations. Developed under contract to NASA, ASDS was created expressly for reuse. The simulations built to date are non-real-time trajectory and vehicle simulations for government customers, both civilian and military. ASDS was designed to permit construction of any conceivable simulation, including real-time.

Typically, fifty to ninety percent of an ASDS application consists of reused parts from the ASDS libraries. ASDS achieves significant reuse through a unique simulation executive structure which implements the primary ASDS strategy: all simulations are characterized by propagation separated by discrete events. This strategy, along with careful design of reusable and integrable parts, allows ASDS users to easily and rapidly prototype high fidelity simulations. The library contains an abundance of mature, generic models, including environment models, vehicle models, events, utilities, and primitives.

MATRIX[®] LIBRARY DEVELOPMENT AND USE

The pilot project of the RDL was a real-time six degree-of-freedom (DOF) simulation of the Soyuz spacecraft when used as the Crew Transfer Vehicle (CTV) for International Space Station Alpha (ISSA). This project involved the significant use of the MATRIX[®] product. [Uhde, 1994] The main goal of the Soyuz/CTV simulation project was to produce a real-time GN&C engineering simulator of the Soyuz/CTV from the deorbit burn through touchdown.

The RDL team took a three phase approach to build the simulation with corresponding versions. Phase 1 built a three DOF simulation with relatively low fidelity models. This allowed the team to learn the toolset and the rapid development process. In Phase 2, the simulation was upgraded to six DOF with substantially higher fidelity models. Phases 1 and 2 used Apollo flight software models and vehicle data, as the Soyuz data was unavailable. Phase 3 incorporated the highest fidelity models deemed necessary, along with the available Soyuz flight software algorithms.

One of the goals of this project was to create a set of libraries of reusable MATRIX[®] parts. These parts allow future projects to leverage off of the Soyuz experience.

The team began by listing all "Utility" type routines expected to be used in the simulation. These mainly featured vector/matrix manipulation routines, coordinate transformations, quaternion routines and orbital elements utilities. Since the team had little previous experience with MATRIX[®], creating these utilities provided hands-on

experience in preparation for the more complicated task of building the simulation. Coding standards were developed during this time and applied to both the utilities and all other simulation models.

A base library of utility functions, such as vector, matrix, and quaternion manipulation procedures was built during Phase 1. Each utility routine (SuperBlock) was fully unit tested and documented. Confidence in the validated utilities helped speed the debugging process when Version 1 of the simulation was integrated.

The simulation size increased by a factor of six for Phase 2. All Phase 1 models were upgraded and many additional models were developed. The entire simulation was reorganized to enhance modularity. Both second and fourth order Runge-Kutta plant model integrators were developed to integrate state parameters. Several new environment models were added. New mass properties, propulsion, and aerodynamics models were developed and tested.

All models are data driven, which makes them as generic as possible and thus suitable as utilities for future projects. The mass properties model, for example, was designed to be compatible with both the Apollo and Soyuz vehicles. The mass properties of a vehicle are input as data on an element by element basis. Dry elements use mass, Center-of-Gravity (CG), and moment of inertia data, while the tank elements require mass, CG, and tank ullage. The model uses these data to compute the mass properties for each vehicle component and combine for the entire vehicle.

Likewise, the propulsion model was designed for the Soyuz vehicle, but since it is data driven, it can be used for any vehicle with one variable thrust bi-propellant main engine and up to 26 bi-propellant Reaction Control System (RCS) jets on the Service Module and 8 mono-propellant RCS jets on the Entry Module. The thrust magnitude, location, direction, specific impulse, and mixture ratio are required data inputs for each RCS jet.

The SuperBlocks considered candidates for library routines were separated out at the end of Phase 2. The team set up four different libraries: Vector/Matrix functions, Quaternion functions, Orbital Elements functions, and a set of Miscellaneous functions. The RDL team saved each function as a separate SystemBuild file, and then all utilities in a set were saved together. Saving the entire set together facilitates loading the set into a developer's simulation.

All utility SuperBlocks have unique names beginning with "UTIL_", so that a developer can tell immediately by looking at his catalog, what utilities he has incorporated into his model. RDL coding standards require a set of "Text Blocks" unique to the utilities which indicate the date and version number and other information. This information was also put in a block within the utility as a comment.

Information placed in a block's comment field appears in the generated code and also in the "Detail Output", whereas text blocks only show in the block diagrams. Other than this information, the utility coding standards match the other SystemBuild standards developed by the lab.

After Phase 3 was completed, three new sets of utilities were generated called Plant_Utils, FSW_Utils and FHW_Utils. Plant_Utils is a set of library routines most likely to be of interest to the Plant side of a simulation. These include environment utilities such as gravity and atmosphere models. The FSW_Utils library contains routines most likely to be of interest to flight software designers such as navigation and guidance routines. The FHW_Utils library contains sensor and effector models. Some of these utilities are actually templates which require modification by the developer specific to an application, rather than stand alone functions.

Brief summaries of currently available RDL utilities in each library are given in the paragraphs that follow. Other utilities are constantly being added. Contact the authors for current information. All of these utilities were developed with MATRIX_X[®] pre Version-4 and as such do not take advantage of the new BlockScript block. Some could probably be re-coded more efficiently using BlockScript.

The Vector Matrix Utility library

This set contains routines such as matrix multiplication, matrix inversion, vector matrix multiplication, and vector unitize and magnitude functions.

The Quaternion Utility Library

This set includes routines such as conversions from transformation matrices to quaternions, normalization, multiplication, rotation, and transformations. Both right and left-handed functions are included.

The Orbital Element Utility Library

This set of routines includes a utility which determines all of 13 orbital elements and utilities to determine individual elements.

The Plant Utility Library

This library contains a variety of environmental models such as atmosphere models, several gravity models, planet relative velocity, bi-propellant engine models, and state integration templates for second and fourth order Runge-Kutta.

The Flight Software Utility Library

This library contains a variety of models often associated with flight software for space vehicles. Included are some navigation routines and state integrators, attitude

propagators, digital filters, and guidance algorithms.

The Sensors/Effectors Utility Library

This library currently contains an Inertial Measuring Unit (IMU) model and an accelerometer model.

The Miscellaneous Utilities Library

This library contains functions such as timers, stopwatches, modulo counters, Sign and Signum functions, and some mathematical functions not provided by the SystemBuild palette.

USE OF LIBRARIES WITH MATRIX_X[®]

MATRIX_X[®] functions well as an integrated software engineering environment. But as a programming language, it lacks many of the capabilities normally expected from other languages such as C or Ada.

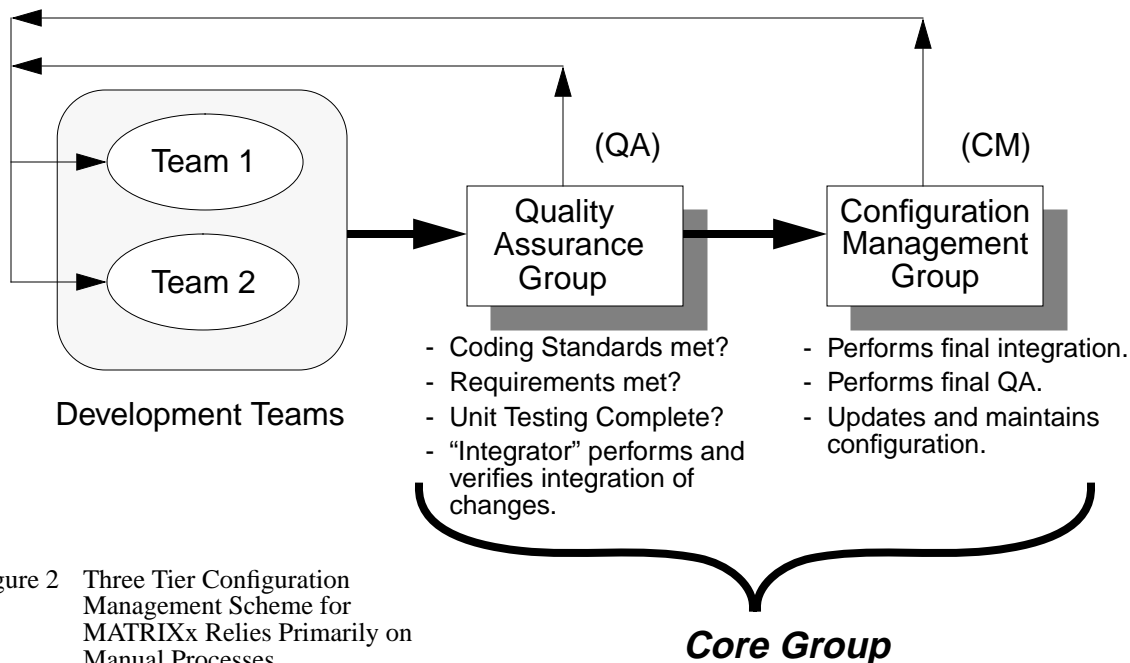
The primary concern is that an entire simulation resides in only one file. An Ada or C programmer normally builds separate logical packages and then links these together to form the simulation. With MATRIX_X[®], the developer must have all blocks in one file before simulating, analyzing, or generating code. This leads to problems with configuration control and library management which are peculiar to MATRIX_X[®].

Programming languages ordinarily take advantage of automated configuration management tools, such as SCCS to allow developers to "check out" specific parts of a simulation. This process inhibits other users from making changes to the checked out portion until it is checked back in. In this sense, libraries of reusable software parts function much like libraries of books. But using libraries with MATRIX_X[®] is akin to a book library with only one, rather large book which may be replicated and modified simultaneously by many users.

Once a utility routine is incorporated into a MATRIX_X[®] simulation, it becomes a local part of that simulation and no longer has a direct relationship to the original library routine. With languages like C or Ada, the libraries are directly linked in after compiling. This guarantees a change in a library routine will get into the simulation. In MATRIX_X[®], changed library routines must be manually inserted into the simulation much like change pages in a paper document.

Incorporating parts of a simulation modified by a developer is a manual operation also, since the developer's file must be manually inserted. A team member, given the role of "integrator", manually selects which SuperBlocks to load and which not to load.

The RDL team handles this problem by using a three tier configuration control process shown in Figure 2. By restricting who can make changes to a configuration



controlled version to only one person, the “integrator”, the RDL team limits the possible sources of error. Each time a change is made and verified by the QA group, the CM group moves it into the configuration managed directories. Each time the configuration controlled version changes, a completely new version is created and date stamped.

This approach suits small projects, skunkworks-type projects, and larger rapid development projects organized around a small core team of experts. But for larger, classically managed projects such as ISSA, these manual techniques are not appropriate. The Space Station Program needs to manage and combine possibly thousands of SuperBlocks from multiple developers, over many years.

“File SuperBlocks” are a new capability available in Version 4. They potentially simplify the creation of reusable libraries of validated parts. Using File SuperBlocks, a developer defines and examines catalogs of libraries different from the current SystemBuild catalog, and loads in a library SuperBlock as a read only block. This capability, solves some of the problems discussed above, but has some problems which limit its usefulness.

These problems are listed in roughly the order of importance.

1. All input and output pin names for a File SuperBlock are invisible to the user. The developer does not know how to hook up a block for other than a trivially simple cases.
2. The developer cannot step into the file block to see what is inside. This makes the block architecture, its

capabilities and limitations, and any relevant comments and text blocks invisible to the developer. Thus, the File SuperBlock permits read only access in only the most restricted sense. Interestingly, if a developer does an interactive simulation, he can step into the file block.

3. Read only capability is too limited. There is no facility to rename the block, making it local and unique. This would be valuable for many reasons. For example, BlockScript utilities need to be encapsulated within SuperBlocks to be accessible as a library. But once in a simulation, a designer may want to expand the block to eliminate a level of hierarchy. Also, many library routines can be considered to be “templates”. For example, BlockScript has the capability of being generic with respect to the number of inputs and outputs, by using `u.size` and `y.size`. A utility could be written process N number of inputs or outputs. The user would load the library block, change the name so that it becomes local, expand the superblock that contains it and change N to what ever he needed.

ASDS OVERVIEW

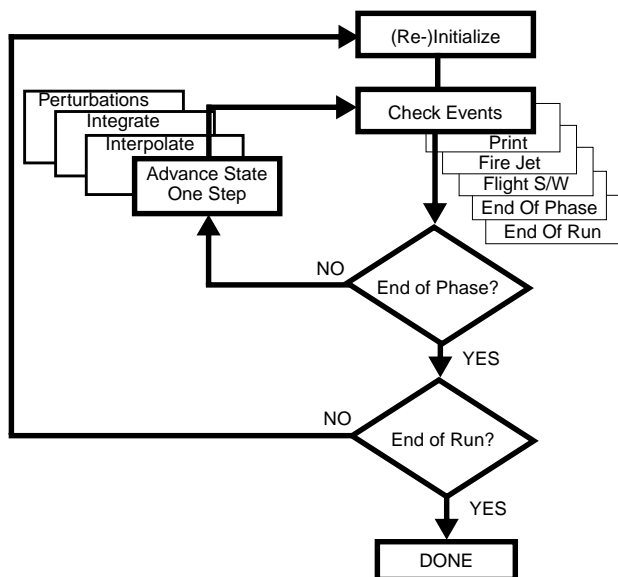
ASDS is a system to facilitate easy and rapid construction of any conceivable simulation. This approach is realized through a library of reusable parts and a strategy to easily assemble them into applications. [Gottlieb, 1994]

The ASDS architectural design is based on the understanding that all simulations consist of propagation

separated by discrete events. The ASDS paradigm insists that nothing discrete is allowed to take place in the propagation function. Allowing discrete events within a propagation step is a common flaw with other simulations which necessarily leads to tight coupling, reduced maintainability, and greatly reduced reusability.

At its highest level, the overall simulation control is captured generically. This concept is shown in Figure 3 below. ASDS implements this concept of propagation separated by discrete events by using four major elements: executive, initialization, discrete events, and propagation. Models, such as environment, vehicle, events, utilities, and primitives, are built and used to support each of these four major elements.

Figure 3 ASDS Simulation Control and Generalized Simulation Driver



Executive

The ASDS executive has two main responsibilities: (1) manage the overall simulation control flow and (2) facilitate communication of necessary information among the four elements of the simulation. Management of the simulation control flow as shown in Figure 3 is contained in the simulation driver. This module contains the flow logic and also defines any unique discrete event functions desired by the user. Information required to be passed among the executive, initialization, discrete event, and propagation elements is contained in a data structure called the *ASDS_Exec* record. This data structure consists of the minimum amount of information necessary to allow proper communication among the generic parts and with the outside world.

Initialization

ASDS uses a powerful input engine to support initialization and re-initialization of simulations. This input capability is based upon the FORTRAN Namelist feature. An Ada version of Namelist, obtained from NASA's Jet Propulsion Laboratory, was enhanced to provide additional features. These include multi-file input sources, units conversion, named phases and datasets, variable-length arrays, multiple data types, variable-string input, user comments, user-defined temporary variables, and equation processing.

ASDS requires each model to be responsible for reading its own data. To support this, data is grouped within named datasets, and that dataset name is known by the model responsible for reading that data. This approach negates the need for centralized input control and distribution of data, and thus avoids the use of global data and increases the modularity of each model.

Discrete Events

A discrete event occurs discretely in a simulation, as opposed to propagation, which is continuous. A discrete event in ASDS can be as simple as writing data to a file or ending a phase, or as complex as executing vehicle flight software.

The ASDS discrete event is comprised of three parts: (1) a function which determines when an action is to be "triggered" (known as the TGO function), (2) the routine(s) that take the action and, (3) the logic for managing the triggers and calling the events.

Propagation

The following describes propagation in ASDS: given a time, determine a state. As such, the method of determining the state becomes inconsequential to the rest of the simulation - discrete events are processed using the state regardless of how it was determined. Therefore, determining the state in ASDS can be done with either integration or interpolation. The uniqueness of this approach means that there is no longer a need for separate executing programs for trajectory generation (integration) and trajectory post-processing (interpolation).

The ASDS library contains many integrators, along with a mature set of high-fidelity perturbation models. The integrators are implemented generically, and the selection of which integrator to use is controlled entirely through user input. Integrators are easily changed during a simulation, the switch is merely a discrete event. For interpolation, the state information is typically provided by an input file.

The ASDS Library of Parts

The ASDS library contains reusable parts to support development of simulations. Experience has shown that these parts typically provide fifty to ninety per cent of the total simulation software required [Neal, 1994]. The remaining simulation software is provided by the developer, and generally consists of vehicle-unique models, unique events (and their associated TGO functions), additional input/initialization, and equations of motion not supplied by ASDS.

New parts are built for a given simulation application, with reuse in mind and subsequently are added to the ASDS library. The library of generic vehicle sensor and effector models grows in this manner.

The ASDS library of reusable parts is divided into 6 sub-libraries: executive, environment models, vehicle models, event models, utilities, and primitives. These sub-libraries contain the code which implements each of the 4 major simulation elements discussed earlier, along with lower level models which support these elements. For example, the environment models sub-library contains perturbation models which support propagation, and the vehicle models sub-library contains vehicle sensor, effector, and flight software models which support the discrete event of vehicle on-board processing.

The ASDS library provides a significant amount of capability with relatively little code. This is a direct consequence of careful design of each module for reuse. Since development, testing, and maintenance costs are typically based directly on the amount of code, this concept of a carefully designed library of reusable parts translates into a significant reduction in software lifecycle expense.

Example ASDS Applications

Applications built with ASDS parts, demonstrate a wide range of domain applicability, rapid prototyping, evolution of verified, flight certified capability, and substantial reuse of parts.

All ASDS-built simulations are inherently multi-vehicle; required memory is dynamically allocated at runtime based upon the user's input of how many vehicles are being simulated. A 3/6/N-DOF simulation "template" contains the integrated simulation framework (i.e., the 4 major elements of the simulation) plus the high fidelity environment perturbations. Known as GENECIS (for *Guidance et Navigation et Control Integrated Simulation*), this template provides the starting point from which specific applications are built.

Since ASDS handles a unique vehicle's processing (sensors, effectors, flight software) as a discrete event, the developer independently builds the vehicle-unique parts. Integration into the simulation is done quickly and efficiently

because GENECIS contains the necessary "hooks" for the addition of vehicle unique models. This approach makes it easy for an ASDS application to accommodate models developed elsewhere, such as encapsulating existing software from other applications or integrating auto-generated code from a graphical software development environment such as MATRIX[®]. Table 1 provides a list of the major applications built using ASDS parts. Many of the applications were built using GENECIS as a starting point, which greatly reduced development time and allowed the developers to concentrate solely on the application's vehicle uniqueness.

Table 1 ASDS-Built Applications

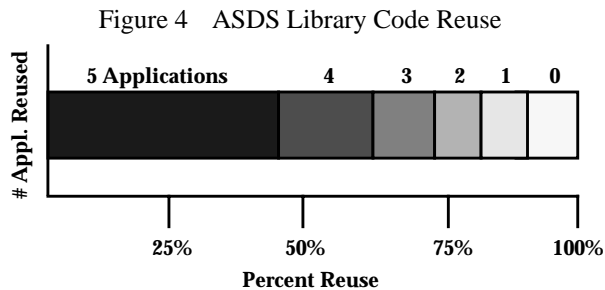
Model	Description	Lang.
BG14	Variation of parameters special perturbation method for propagation	Ada, C
GENECIS	3/6/N-DOF environment simulation - template for creating vehicle-unique sims	Ada
STARS*	Space Station 6-DOF reboost / 3-DOF decay (using BG14)	Ada
ARCSIM*	6-DOF multi-vehicle automated rendezvous & capture (AR&C)	Ada, C, C++, Fortran
Threat-Sim*	Threat missile 3-DOF powered flight, 6-DOF coasting flight	Ada
SMAARTS	3-DOF Montecarlo aerobraking	Ada
LunEx	3/6-DOF Earth orbit-to-Moon landing	Ada
R3BTARG/ R3BHALO	Restricted 3 body targeting for libration point missions & weak stability boundary applications	Ada
NEAR-Tool	Converged, optimized high precision trajectory sim for asteroid rendezvous & orbit	Ada
Fast Phi	Unrestricted 4-body optimization	Ada
Space Shuttle 6-DOF	Ascent, on-orbit, and descent mission planning simulations	C
Generic 3-DOF	Generalized multi-vehicle 3-DOF	C

* Began with GENECIS simulation framework

Figure 4 shows the amount of reuse provided by the ASDS library for the first five applications in Table 1. Three of the five applications reuse greater than seventy per cent of the library source lines, determined by Ada semi-colons. All five of the applications reuse nearly half of the library code. Figure 5 gives, for each application, the amount of reused library code versus new application code.

REUSING DOCUMENTATION

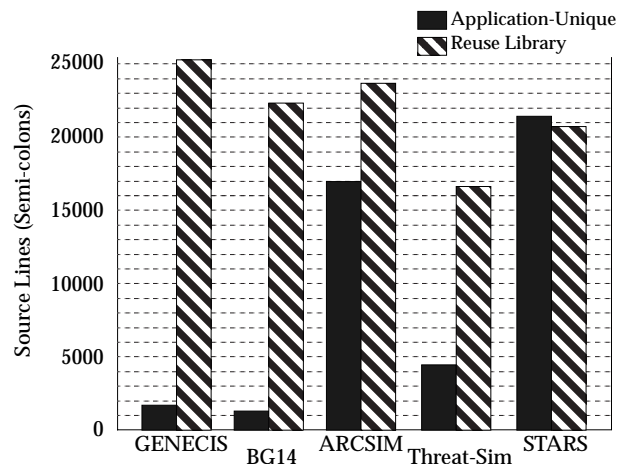
ASDS documentation applies the same reusable parts approach taken with software production. ASDS and its applications are documented using FrameMaker¹ and its



book feature, which permits logically separate entities (i.e., separate ASDS models and functions) to be documented separately. The flexibility offered by this approach allows quality documentation to be generated and reused for several different applications, with less cost and less effort.

Existing ASDS documentation includes formulation guides, user guides, and standards manuals for both ASDS in general and specific applications. These documents are available in both paper and on-line (electronic, using hypertext) form. Additional documents for specific applications are prepared as required by customers, including requirements specification, design documents, and verification reports. ASDS documentation can support both

Figure 5 ASDS Application Reuse of Library



MIL-STD-2167A and NASA-STD-2100-91 documentation standards.

COMBINING ASDS WITH MATRIXx

ASDS and MATRIXx[®] offer unique capabilities, both of which are required for complete, end-to-end flight software and vehicle development. MATRIXx[®]'s strengths are in the real-time flight code development environment. ASDS offers

1. FrameMaker is a registered trademark of Frame Technology Corporation

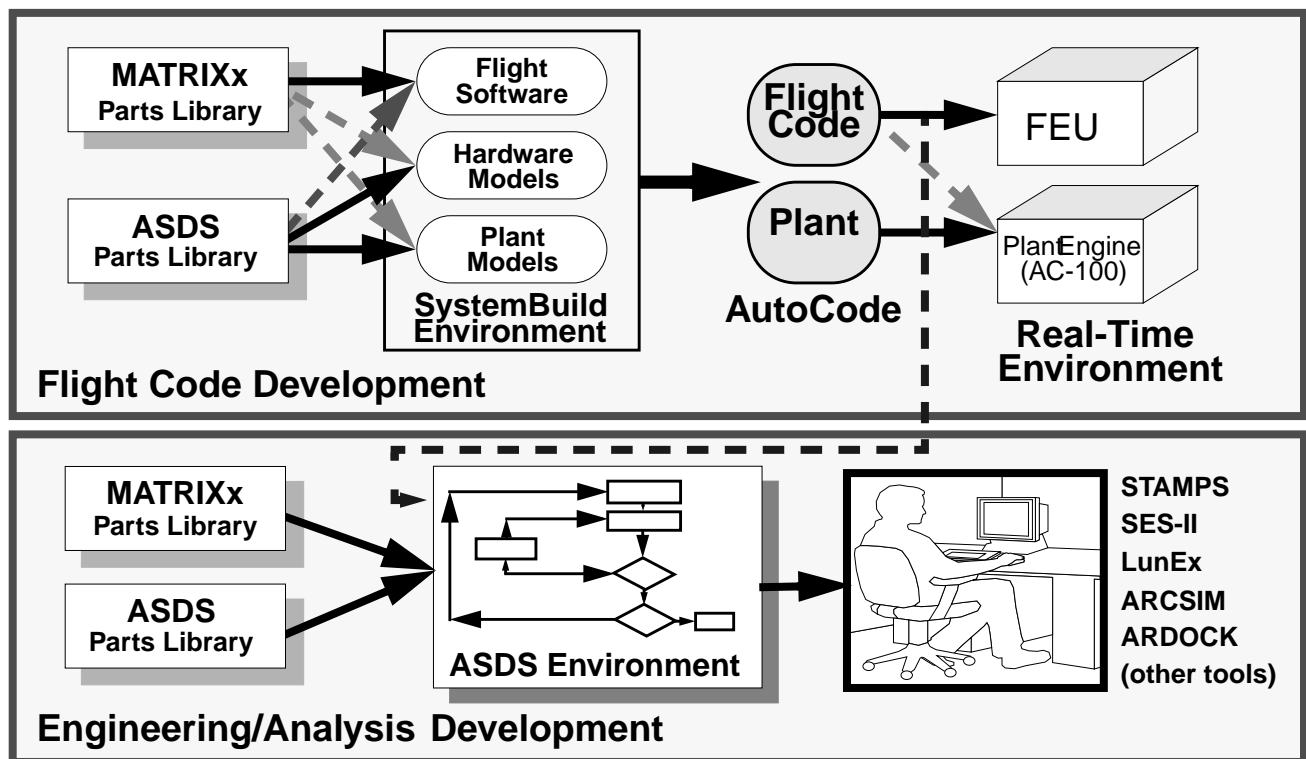


Figure 6 ASDS and MATRIXx Paths to Software Development and Analysis

a sophisticated engineering simulation and tool development environment.

The software development paths preferred by the RDL combine MATRIX_X[®] and ASDS in a way that takes advantage of the strengths and capabilities of each. This combined approach is shown in Figure 6. MATRIX_X[®] is the primary component on the path to the real-time flight code development. ASDS provides the more complex functions that are difficult or impracticable to build in MATRIX_X[®] or functions that have been previously built and validated and need not be duplicated in MATRIX_X[®].

The lower part of the figure shows the path for non-real-time tool and analysis. ASDS typically provides much higher fidelity environment models and improved data capture and processing. This is advantageous when the plant models built in MATRIX_X[®] are lower fidelity versions simplified for real-time execution, or when faster than real-time is required. The flight code model comes directly from the MATRIX_X[®] real-time code, and this is installed and executed with the ASDS plant. This transition requires changes to the AutoCode[™] template and possibly to the User Code Blocks which handle hardware specific functions.

Individual models previously developed and validated in MATRIX_X[®], such as an actuator model, are provided to the ASDS libraries as procedures. This is accomplished via auto-generating code through ASDS-compatible templates

CONCLUSIONS

Library reuse within the MATRIX_X[®] software engineering environment can be significant and successful for projects using small core teams or skunkworks-type operations. Larger more classically managed projects such as ISSA will need automated approaches to library management. Automated configuration management tools designed for traditional languages like C or Ada are not well suited for use with MATRIX_X[®], and the MATRIX_X[®] product currently has no tools suited for configuration and library management.

ASDS is designed expressly to take advantage of software reuse, and has demonstrated considerable success with this design approach. Configuration management and library management can take advantage of current tools commonly available.

REFERENCE

Bordano, A.; Uhde-Lacovara, J.; DeVall, R.; Partin, C.; Sugano, J.; Doane, K.; Compton, J. "Cooperative GN&C Development In A Rapid Prototyping Environment". In Proceedings of the AIAA Computing in Aerospace 9 Conference (San Diego, CA, Oct. 19-21, 1993). AIAA,

Washington, D.C., AIAA-93-4622-CP, 883-890

Uhde, J.; Weed, D.; McCleary, B.; Wood, R. "The Rapid Development Process Applied to Soyuz Simulation Production". In Proceedings of the European Simulation Multi-Conference (Barcelona, Spain, June 1-3, 1994). pp1071, Society for Computer Simulation International, ISBN 1-56555-028-5

Gottlieb, R.; Neal, S. D. "The Advanced Simulation Development System (ASDS)". In proceedings of the Society for Computer Simulation (SCS) Summer Computer Simulation Conference (San Diego, July 18-20, 1994).

Neal, S. D. 1994. "Advanced Simulation Development System (ASDS)- Demonstration of Capability", MDA-TM-IRAD94-01, Houston, Tx. (Mar.).

ACKNOWLEDGMENTS

The authors wish to acknowledge the following people without whose work this project could not have proceeded:

- NASA-JSC: John Craft, Jr., John Ruppert, Bruce Schulz
- MDA-HD: Stan Fernandes, Bret McCleary
- MDA-HB: James Ball, Heiko Jones, John Riel

BIOGRAPHIES

Dr. Jo Uhde is the manager of the RDL at NASA-JSC. She received her Ph.D. in electrical engineering from Stevens Institute of Technology, Hoboken, NJ, in 1984, as well as her Master's of Engineering, Electrical in 1977, and her Bachelor's of Engineering in 1975. She was an instructor and assistant professor of electrical engineering at Stevens from 1980 to 1990. She was appointed four times as a NASA/ASEE Summer Faculty Fellow at NASA-JSC. She is a member of Tau Beta Pi, Eta Kappa Nu and Sigma Xi.

Daniel Weed is a Senior Engineer with McDonnell Douglas Aerospace in Houston, TX. He received his BS in Aerospace Engineering from the University of Texas at Austin in 1984. Since then he has been involved in simulation modeling, crew procedures development, and flight software development, supporting programs at the Johnson Space Center including the Space Shuttle, the Space Station, and Space Exploration Initiative projects.

Robert G. Gottlieb holds BS and MS degrees in Mechanical Engineering from the Massachusetts Institute of Technology, and a Ph.D. in Aerospace Engineering from the University of Texas at Austin. Dr. Gottlieb has been involved in simulation modeling and development for more than 25 years, supporting various Army programs and the Apollo, Skylab, Space Shuttle, and Space Station programs. He is the original developer of ASDS, and has designed and developed most of its key features. He holds several patents, and has been recognized by NASA for several new technology

disclosures.

Douglas Neal holds a BS degree in Aerospace Engineering from Iowa State University, and is a member of Phi Kappa Phi and Sigma Gamma Tau honor societies. He has been involved in software and simulation development for the past 12 years, and has led the Advanced Simulation Development System effort for the past 3 years, working most recently to develop object-oriented library parts using C++.

THE RAPID DEVELOPMENT PROCESS APPLIED TO SOYUZ SIMULATION PRODUCTION

Jo Uhde-Lacovara Ph.D.
NASA Johnson Space Center
Houston, Texas 77058

Daniel Weed, Bret McCleary, Ron Wood
McDonnell Douglas Aerospace - Space Systems
13100 Space Center Boulevard.
Houston, Texas 77059

ABSTRACT

The Navigation, Control & Aeronautics Division (NCAD) at the National Aeronautics and Space Administration-Johnson Space Center (NASA-JSC) is exploring ways of producing Guidance, Navigation & Control (GN&C) flight software faster, better and cheaper. To achieve these goals NCAD established hardware/software facilities to take an avionics design project from initial inception through high fidelity real-time, hardware-in-the-loop (HIL) testing.

This paper concentrates on the use of commercial, off-the-shelf (COTS) software products to develop the GN&C algorithms in the form of graphical data flow diagrams, to automatically generate source code from these diagrams and to run in a real-time, HIL environment under a rapid development paradigm.

To evaluate these concepts and tools, NCAD embarked on a project to build a real-time, six degree-of-freedom (DOF) simulation of the Soyuz Assured Crew Return Vehicle (ACRV) flight software (FSW). To date, a productivity increase of 50% has been seen over traditional NASA methods for developing engineering simulations.

INTRODUCTION

Current fiscal realities demand that GN&C simulations, requirements and FSW be developed faster, cheaper and without any loss to quality. NCAD is exploring new approaches and processes for the creation of these products that will significantly reduce space vehicle design costs. Specific goals for this initiative are as follows:

- 1) Identify appropriate commercial software technologies.
- 2) Demonstrate a subset of these technologies on selected space vehicle programs.
- 3) Benchmark cost/schedule performance against past programs.

TRADITIONAL SOFTWARE DESIGN APPROACH

The traditional FSW development approach starts with a requirements design phase in which engineers develop and test candidate GN&C algorithms. A non-real-time engineering simulation is created in which the performance of the candidate algorithms is compared. During this phase, several reviews are scheduled which result in some of the algorithms being dropped from further consideration. Once the algorithms are selected, a requirements document is written from which a separate group of software engineers writes the FSW. This process is illustrated in Figure 1.

While the GN&C algorithms are being selected, avionics engineers select the on-board computers and data bus architecture using preliminary estimates of the number of lines of GN&C software required. Because the on-board computers have limited capabilities, scheduling the FSW so that it runs in real-time and produces the required results is a difficult process. Real-time simulations are created to aid in the real-time development.

This process becomes very costly when changes to the vehicle requirements are made. For example, a change in the mission requirements may force a change in the FSW requirements. These changes can require extensive modifications to the FSW. This necessitates changes in both the non-real-time and real-time simulations to support the effort.

THE RAPID DEVELOPMENT PROCESS

An improved approach to developing real-time FSW is to produce prototype real-time simulations and code concurrently with the requirements development. Once the first working prototype of the FSW is developed, HIL testing is initiated.

This process allows incompatibilities in the software design, implementation, or hardware selection to be discovered early in the development. This cycle of concurrent requirements and software development, and HIL testing is repeated until acceptable software is produced. The

approach used here is a “spiral” development approach where developers “build a little, test a little”. This is illustrated in Figure 2.

THE RAPID DEVELOPMENT LABORATORY

In an effort to evaluate the rapid development process and tools, NCAD created the Rapid Development Laboratory (RDL). The RDL is a JSC on-site resource dedicated to exploring and evaluating new technologies and processes for FSW and simulation development. In order to assess these techniques, the RDL team members embarked on a pilot project to build a real-time, six DOF simulation of the Soyuz/ ACRV FSW (Bordano 1993).

The project is a teaming effort between personnel from NASA-JSC, Lockheed Engineering Services Corporation (LESC) and McDonnell Douglas Aerospace in Houston, TX (MDA-HD). A cooperative agreement allowed engineers at JSC to access a similar facility at the McDonnell Douglas Aerospace site in Huntington Beach, CA (MDA-HB), where development for the Delta Clipper (DC-X) was performed.

The RDL team used the MATRIX_X[®]/SystemBuild[™] product line available from Integrated Systems Inc. (ISI),

Santa Clara, CA, to build the entire simulation in block diagram form. Figure 3 shows the application of ISI’s toolset to the spiral development process.

MATRIX_X[®]/SystemBuild[™] is a graphical software tool which allows the user to develop data flow block diagrams of the desired system using available primitives from a palette. These elementary blocks are organized in groups called “Superblocks” which become procedures or subtasks. This leads to highly modular software designs well suited to the development of generic software libraries and software reuse. Once the software data flow diagrams are built and linked together, they are interactively tested in a non real-time environment. Time and frequency domain analysis are also performed interactively.

Real-time code is automatically generated using the AutoCode[™] tool to produce source code from the block diagram representation in FORTRAN, C, or Ada. Legacy code may be imported into the model via User Code Blocks. The source code is compiled and run on the AC-100[™] real-time computer to verify real-time and HIL performance.

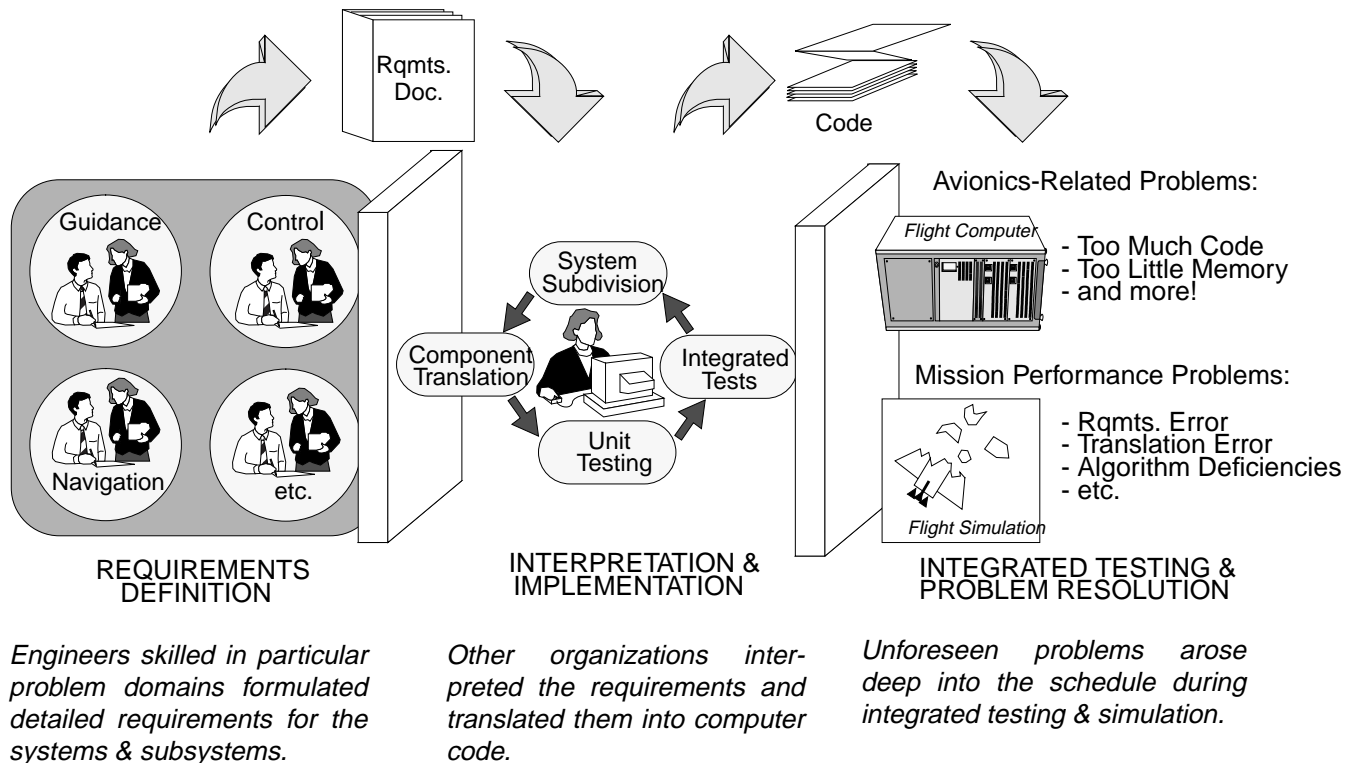


Figure 1: The Traditional Flight Software Development Approach

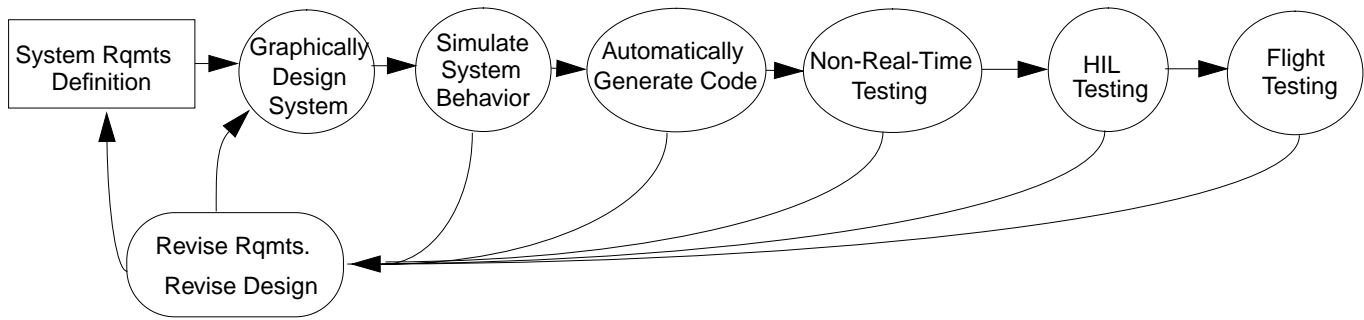


Figure 2: The Spiral Development Process

THE SOYUZ/ACRV SIMULATION

The main goal of the Soyuz/ACRV simulation project is to produce a real-time GN&C engineering simulator of the Soyuz/ACRV from the deorbit burn through touchdown. The Soyuz/ACRV mission profile is shown in Figure 4.

The team took a three phase approach to build the simulation. Phase 1 built a three DOF simulation with elementary models which allowed the team to learn the toolset and process. In Phase 2, the simulation was upgraded to six DOF with substantially higher fidelity models. Phases 1 and 2 used Apollo flight software models and vehicle data as the Soyuz data was unavailable. Phase 3 will incorporate the highest fidelity models deemed necessary, along with the actual Soyuz flight software algorithms. This phase is in progress.

The first executable version was built in a matter of days. Although it had very crude models, it allowed the team to exercise all steps of the rapid development process short of flight testing. A basic library of utility functions, such as vector, matrix, and quaternion manipulation procedures, was also built during this phase. The first working version used the “build a little, test a little” philosophy to build new and more complex models, testing the entire integrated product, in a real-time environment each step of the way.

The Phase 1 simulation was developed using simple environment models such as a constant coefficient (C_L , C_D) aerodynamic model, an exponential atmosphere model, and a central force gravity model.

The FSW models developed for Phase 1 included the most basic guidance, navigation, and flight control algorithms. The deorbit guidance routine assumed an initial circular orbit and targeted a new orbit with a specified target perigee. The entry guidance was open-loop and steered to a constant bank angle. The angle-of-attack and sideslip angles were set to constant values to simplistically model aerodynamic trim. A perfect navigation model was used in

the flight software. A choice of perfect or filtered attitude control was provided in the flight control module. The filter was a second order digital filter with rate and acceleration limiting.

The simulation size increased by a factor of six for Phase 2. All Phase 1 models were upgraded and many additional models were developed. The entire simulation was reorganized to enhance modularity. The simulation was redesigned so that all mission constants, vehicle data, and environmental parameters were read from input files. Both second and fourth order Runge-Kutta plant model integrators were developed to integrate state parameters.

Two environment models were added: a new atmosphere model and a new gravity model. The simulation accommodated all atmospheric density and speed of sound data that are a monotonic function of altitude. The simulation read in all atmospheric data from a file. A 4x0 gravity model was developed and incorporated into the Phase 2 plant model.

New mass properties, propulsion, and aerodynamics models were developed, tested, and implemented in Phase 2. All the models were data driven. The mass properties model was designed to be compatible with both the Apollo and Soyuz vehicles with the mass properties of a vehicle input on an element by element basis. Dry elements used mass, Center-of-Gravity (CG), and moment of inertia data while the tank elements required mass, CG, and tank ullage. This data was used to compute the mass properties for each vehicle component and the entire vehicle.

The propulsion model was designed for the Soyuz vehicle, but can be used for any vehicle with one variable thrust bi-propellant main engine and up to 26 bi-propellant Reaction Control System (RCS) jets on the Service Module and 8 mono-propellant RCS jets on the Entry Module. The thrust magnitude, location, direction, specific impulse (I_{sp}), and mixture ratio are required data inputs for each RCS jet.

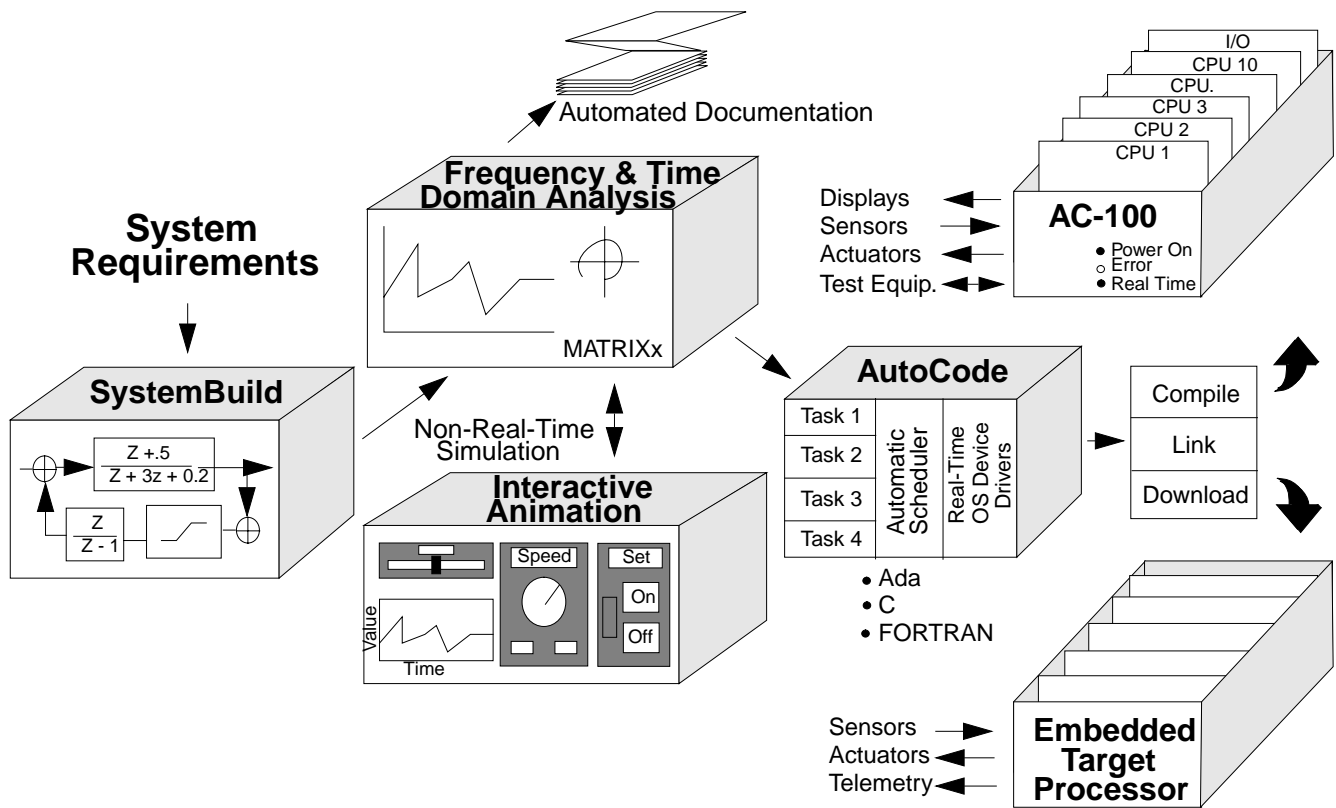


Figure 3: The Application of The ISI Toolset to Spiral Development

The aerodynamics models included a vehicle model and two parachute models. The vehicle model defined aerodynamic coefficients as a function of Mach number, angle-of-attack, and sideslip angle. Rate damping coefficients were also included. The parachute models deployed the parachutes in reefed stages to realistically model the increase in drag during parachute deployment.

Navigation user parameter processing was developed to compute deorbit and entry guidance parameters. A closed-loop deorbit guidance as well as the Apollo entry guidance and flight control systems were developed, tested, and implemented.

The Soyuz 6-DOF simulation was tested in real-time using the AC-100™. The AC-100™ hardware contains digital, analog, serial, and ethernet input and output capability. The AC-100™ package contains the all software needed to generate, compile, link, and run user code on the processor. The simulation was monitored via interactive animation (IA), an AC-100™ tool which allows the user to quickly develop real-time displays using icons such as strip charts, digital output, push-buttons, and sliders.

The simulation was run in several different configurations including a single i860 processor, a single

80386 processor, and the plant model running on the i860 and the FSW running on the 80386. All tests showed no frame overrun errors. Inputs from or outputs to external hardware can be easily defined through the hardware connection editor (HCE) of the AC-100™. A rotational hand-controller was connected to the AC-100™, and the developers exercised the simulation by providing real-time inputs to the flight control system.

Several 3-dimensional vehicle animation displays were developed to enhance the simulation demonstration. To minimize the development cost, many of the Soyuz graphics were obtained from other ACRV or Space Station projects. The displays were animated using the Tree Display Manager (TDM) running on a Silicon Graphics workstation. The displays were driven by simulation data output in real-time from the AC-100™. The data was passed from the AC-100™ to the Silicon Graphics workstation using a serial data communications line. Software was written to extract the data from the serial port and load it into TDM buffers.

SIMULATION PROJECT METRICS

Table 1 shows each of the Phase 2 metrics compared against the Phase 1 metrics. The results show that

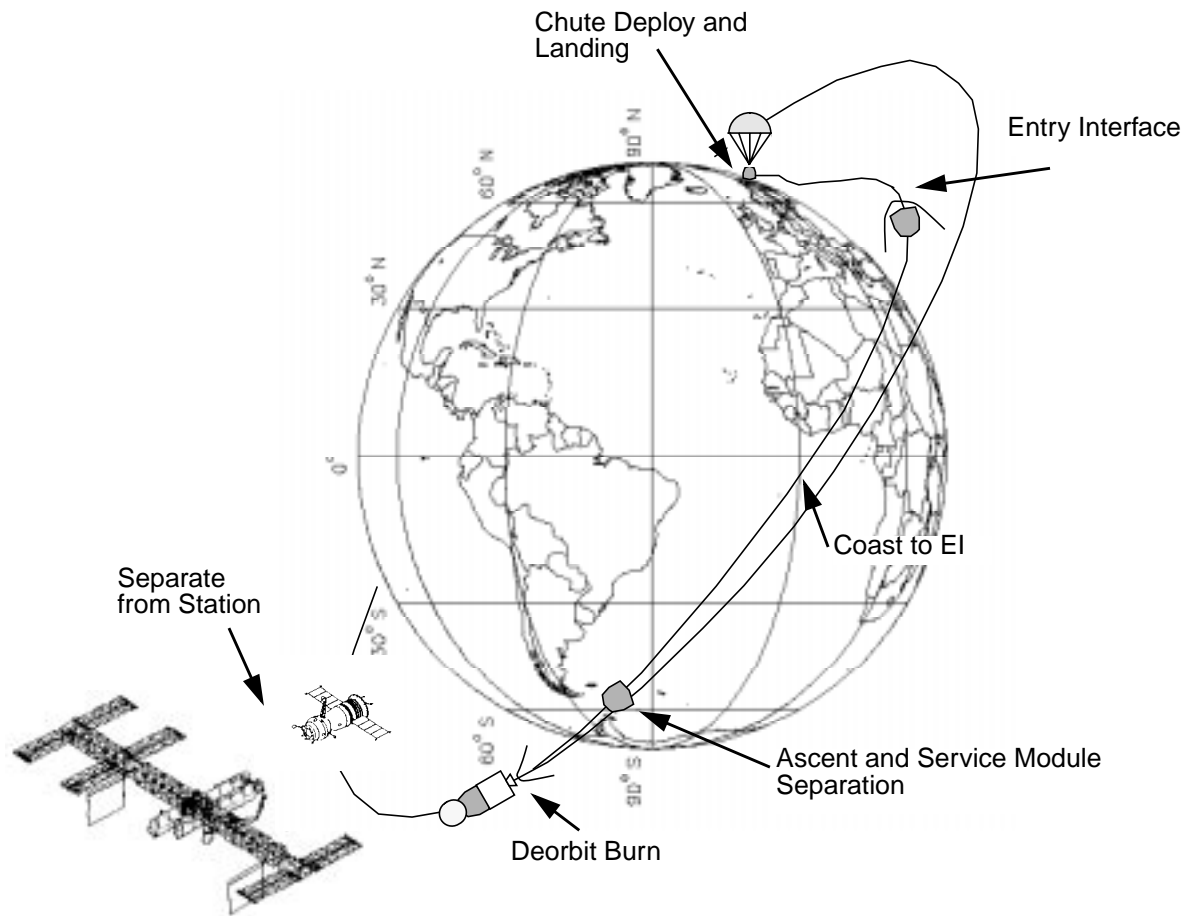


Figure 4: Soyuz Simulation Trajectory Modeling

approximately 22 source lines of code (SLOC) were developed per staff-day. This is a slight improvement from the Phase 1 productivity numbers. Since Phase 1 included much time spent on training, developing the process, building reuse libraries, and initial bug resolution, a larger increase in productivity was anticipated, from Phases 1 to 2.

The COConstructive COSt Model (COCOMO) was used to produce an estimate of the number of staff-hours required to complete the same project with traditional software development approaches. The COCOMO model estimated that 3400 and 11658 staff-hours should be required to produce Phases 1 and 2 respectively. The productivity as compared to the COCOMO model dropped from Phase 1 to Phase 2.

This loss of productivity was primarily caused by several bugs in the ISI tool. Problems were also encountered when the simulation became too large to be supported by the production release of the tool. The RDL team established a process for refining discovered bugs down to simple test cases, where possible, and then passing these along to ISI

personnel. This relationship with the vendor allowed for the bugs to be resolved, and for the capabilities of the tool to be expanded.

LESSONS LEARNED

A summary of some of the lessons learned throughout the Soyuz simulation development are given below:

- A small, well-trained, and diverse team is most efficient when following the rapid development process.
- An executive group should be organized at project start-up. This group should be responsible for common model development, testing, documentation, and configuration control.
- A set of strict, but minimal, coding standards results in a more homogeneous product. Configuration control and unit testing procedures should be defined early in the project. A team should be set up to review all models before integration.
- The coding standards and sharing of common models should be integrated across multiple rapid development projects.

Table 1: Soyuz/ACRV Simulation Phase 1 and 2 Metrics

	Phase 1	Phase 2
Number of Superblocks	55	371
Number of SLOC	4102	25045
Estimated Total Staff-Hours	1830	7720
SLOC per staff-day (assumes re-use of Phase 1 code)	18	22
Approximate Productivity (Actual Vs. COCOMO)	185%	150%
Productivity Increase	85%	50%

- Unit test every model. It is very difficult when using MATRIX_X[®] to find an error during integrated testing. The unit testing procedure should require that expected results from the test be documented before the test is completed. To minimize the integration testing time, all models should be tested in the baseline simulation before being formally integrated.
- Real-time issues become readily apparent when using the MATRIX_X[®] tool. The software design can be modified early in the project to account for real-time design issues.
- A large amount of time is spent in finding workarounds for MATRIX_X[®] bugs. Allocate a percentage of project development time to MATRIX_X[®] bug resolution. Some bugs are difficult and time-consuming to find and fix, or provide workarounds for.
- Many enhancements to the MATRIX_X[®] tool are needed. Some desirable features of modern programming languages are not available yet. The Soyuz/ACRV simulation stretched the MATRIX_X[®] tool to its limits. The tool must grow in order to support very large-scale simulations.

CONCLUSIONS AND FUTURE PLANS

One of the main lessons learned from this project is that domain experts can develop GN&C software with significant productivity increases, yet work in their native language, that of the block diagram. However, the toolset, while a good first step in the rapid design of GN&C FSW is not completely mature. Many enhancements are needed before these tools are optimized for development of GN&C systems. This project has pushed the toolset to its limits due to both the size and complexity of the model. This has led to a good working relationship with the vendor as the toolset is refined.

There is interest in using the Soyuz/ACRV simulation as the basis for a mid-fidelity trainer for flight crew and mission operations personnel at NASA-JSC. The RDL has also been proposed as the hub of a distributed simulation environment for early integration of FSW and hardware elements for

International Space Station Freedom. The RDL supports a Mosaic exhibit on the NASA-JSC home page. There are plans to link this to an anonymous File Transfer Protocol (FTP) site to allow model and utility sharing with other users of MATRIX_X[®].

REFERENCE

Bordano, A.; Uhde-Lacovara, J.; DeVall, R.; Partin, C.; Sugano, J.; Doane, K.; Compton, J. "Cooperative GN&C Development In A Rapid Prototyping Environment." In Proceedings of the AIAA Computing in Aerospace 9 Conference (San Diego, CA, Oct. 19-21, 1993). AIAA, Washington, D.C., AIAA-93-4622-CP, 883-890

ACKNOWLEDGMENTS

The authors wish to acknowledge the following people without whose work this project could not have proceeded:

NASA-JSC: John Craft, Jr., John Ruppert, Bruce Schulz

MDA-HD: Stan Fernandes, Joo Ahn Lee, Brian Rishikof, Lou Zyla

MDA-HB: James Ball, Heiko Jones, John Riel

LESC: Mike Gulizia, Kent Kaiser, Ron Smith, Marv Walton

BIOGRAPHY

JO UHDE-LACOVARA is the manager of the RDL at NASA-JSC. She received her Ph.D. in electrical engineering from Stevens Institute of Technology, Hoboken, NJ, in 1984, as well as her Master's of Engineering, Electrical in 1977, and her Bachelor's of Engineering in 1975. She was an instructor and assistant professor of electrical engineering at Stevens from 1980 to 1990. She was appointed four times as a NASA/ASEE Summer Faculty Fellow at NASA-JSC. She is a member of Tau Beta Pi, Eta Kappa Nu and Sigma Xi.

**We are terribly sorry, but
this document is not yet
available on-line.**

**We apologize for the
inconvenience.**